Deklarative Verarbeitung von Datenströmen in Sensornetzwerken

Dissertation
Zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt der
Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von
Dipl.-Inf. Daniel Klan

Tag der Einreichung: 15. September 2010
Tag der wissenschaftlichen Aussprache: 22. Februar 2011

Gutachter

1. Prof. Dr.–Ing. Kai-Uwe Sattler
2. Prof. Dr.–Ing. Bernhard Seeger
3. Prof. Dr.–Ing. Wolfgang Lehner

urn:nbn:de:gbv:ilm1-2011000046

Für Michael und Roland

Zusammenfassung

Sensoren finden sich heutzutage in vielen Teilen des täglichen Lebens. Sie dienen dabei der Erfassung und Überführung von physikalischen oder chemischen Eigenschaften in digital auswertbare Größen. Drahtlose Sensornetzwerke als Mittel zur großflächigen, weitestgehend autarken Überwachung von Regionen oder Gebäuden sind Teil dieser Brücke und halten immer stärker Einzug in den industriellen Einsatz. Die Entwicklung von geeigneten Systemen ist mit einer Vielzahl von Herausforderungen verbunden. Aktuelle Lösungen werden oftmals gezielt für eine spezielle Aufgabe entworfen, welche sich nur bedingt für den Einsatz in anderen Umgebungen eignen. Die sich wiederholende Neuentwicklung entsprechender verteilter Systeme sowohl auf Hardwareebene als auch auf Softwareebene, zählt zu den wesentlichen Gründen, weshalb entsprechende Lösungen sich zumeist im hochpreisigen Segment einordnen. In beiden Entwicklungsbereichen ist daher die Wiederverwendung existierender Module im Interesse des Anwenders. Stehen entsprechende vorgefertigte Lösungen bereit, besteht weiterhin die Aufgabe, diese in geeigneter Form zu kombinieren, so dass den vom Anwender geforderten Zielen in allen Bereichen genügt wird. Insbesondere im Kontext drahtloser Sensornetzwerke, bei welchen mit stark beschränkten Ressourcen umgegangen werden muss, ist für das Erzeugen passender Lösungen oftmals Expertenwissen von Nöten.

Im Mittelpunkt der vorliegenden Arbeit steht die energie-effiziente Datenanalyse in drahtlosen Sensornetzwerken. Hierzu wird mit *AnduIN* ein System präsentiert, welches den Entwurf auf Softwareebene dahingehend vereinfachen soll, dass der Anwender lediglich die Aufgabenstellung unter Verwendung einer deklarativen Anfragesprache beschreibt. Wie das vom Anwender definierte Ziel erreicht wird, soll vollautomatisch vom System bestimmt werden. Der Nutzer wird lediglich über die Definition gewünschter Eigenschaften in den Entwicklungsprozess integriert.

Die dauerhafte Überwachung von Objekten mittels drahtloser Sensornetzwerke hängt von einer Vielzahl von Parametern ab. Es hat sich gezeigt, dass insbesondere der Energieverbrauch bei der drahtlosen Datenübertragung eine der wesentlichen Schwachstellen ist. Ein möglicher Ansatz zur Reduktion des Energiekonsums ist die Verringerung des Kommunikationsaufwands aufgrund einer frühzeitigen Auswertung von Messergebnissen bereits auf den Sensorknoten. Oftmals ist eine vollständige Verarbeitung von komplexen Algorithmen im Sensornetzwerk aber nicht möglich bzw. nicht sinnvoll. Teile der Verarbeitungslogik müssen daher auf einer zentralen Instanz ausgeführt werden. Das in der Arbeit entwickelte System integriert hierzu sowohl einfache als auch komplexe, nur teilweise im Sensornetzwerk verarbeitbare Verfahren. Die Entscheidung, welche Teile einer Applikation auf welcher Komponente ausgeführt werden, wird vom System selbstständig auf Basis eines mehrdimensionalen Kostenmodells gefällt.

Im Rahmen der Arbeit werden weiterhin verschiedene Verfahren entwickelt, welche insbesondere im Zusammenhang mit der Analyse von Sensordaten von Interesse sind. Die erweiterten Algorithmen umfassen Methoden zur Auswertung von Daten mit räumlichem Bezug, das Data Cleaning mittels adaptiver Burst-Erkennung und die Identifikation von häufigen Mustern über quantitativen Itemsets.

Abstract

Sensors can now be found in many facets of every day life, and are used to capture and transfer both physical and chemical characteristics into digitally analyzable data. Wireless sensor networks play a central role in the proliferation of the industrial employment of wide-range, primarily autonomous surveillance of regions or buildings. The development of suitable systems involves a number of challenges. Current solutions are often designed with a specific task in mind, rendering them unsuitable for use in other environments. Suitable solutions for distributed systems are therefore continuously built from scratch on both the hardware and software levels, more often than not resulting in products in the market's higher price segments. Users would therefore profit from the reuse of existing modules in both areas of development. Once prefabricated solutions are available, the remaining challenge is to find a suitable combination of these solutions which fulfills the user's specifications. However, the development of suitable solutions often requires expert knowledge, especially in the case of wireless sensor networks in which resources are limited.

The primary focus of this dissertation is energy-efficient data analysis in sensor networks. The *AnduIN* system, which is outlined in this dissertation, plays a central role in this task by reducing the software design phase to the mere formulation of the solution's specifications in a declarative query language. The system then reaches the user's defined goals in a fully automated fashion. Thus, the user is integrated into the design process only through the original definition of desired characteristics.

The continuous surveillance of objects using wireless sensor networks depends strongly on a plethora of parameters. Experience has shown that energy consumption is one of the major weaknesses of wireless data transfer. One strategy for the reduction of energy consumption is to reduce the communication overhead by implementing an early analysis of measurement data on the sensor nodes. Often, it is neither possible nor practical to perform the complete data analysis of complex algorithms within the sensor network. In this case, portions of the analysis must be performed on a central computing unit. The *AnduIN* system integrates both simple methods as well as complex methods which are evaluated only partially in network. The system autonomously resolves which application fragments are executed on which components based on a multi-dimensional cost model.

This work also includes various novel methods for the analysis of sensor data, such as methods for evaluating spatial data, data cleaning using burst detection, and the identification of frequent patters using quantitative item sets.

Danksagung

Es gibt viele Menschen, denen ich für ihre Unterstützung bei der Erstellung dieser Arbeit Dank schulde. An erster Stelle ist dies Kai-Uwe Sattler, der mich mit zahllosen Ratschlägen und seiner schier endlosen Geduld immer wieder dem erfolgreichen Abschluss dieser Arbeit ein Stück näher gebracht hat. Ohne seine Unterstützung wäre diese Arbeit sicherlich nicht möglich gewesen. Ebenfalls bedanken möchte ich mich bei den beiden Gutachtern Wolfgang Lehner und Bernhard Seeger, welche sich Zeit für mich und meine Arbeit genommen haben.

Weiterhin möchte ich mich bei meinen Kollegen für das gute Arbeitsklima bedanken. Viele von ihnen sind mir über die Jahre zu Freunden geworden und haben mir gezeigt, dass Ilmenau auch jenseits der Universität ein Leben bietet. Mein besonderer Dank gilt dabei Marcel Karnstedt, der lange Zeit mit mir das Zimmer teilte und jederzeit für Fragen und Probleme offen war. Außerdem möchte ich mich bei Francis Gropengießer, Elizabeth und Stephan Baumann, sowie Cordula Giewald bedanken. Ihr wisst schon warum.

Weiterhin danke ich Martin Sauerbrey und Volker Neundorf. Martin hat mir geholfen alle Anpassungen an den verwendeten Sensorknoten vorzunehmen und damit erst den entwickelten Prototypen möglich gemacht. Volker hat mich in die Grundlagen der E-Technik einweiht und mir gezeigt, dass Ströme nicht nur aus Bits und Bytes bestehen sondern auch in Kabeln fließen und mit Oszilloskopen gemessen werden können.

Ich möchte mich zudem bei allen Studenten bedanken, deren Studien-, Diplom- und Projektarbeiten ich betreut habe und welche aktiv an der stetigen Verbesserung von AnduIN beteiligt waren. Besonderer Dank gilt dabei Thomas, Timo, Stefan, Felix und Christian.

Zuletzt möchte ich den beiden wichtigsten Menschen in meinem Leben danken, meiner Frau und meiner Tochter. Eine Dissertation kostet viel Zeit und hinterlässt ihre Spuren. Bei euch habe ich jederzeit Rückhalt gefunden. Danke für alles.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

C	Verarbeitungskosten auf der zentralen Komponente
E	Energiekosten im WSN
$\mathbf{O}$	Menge aller logischen Operatoren
$\mathcal{L}$	vom DSMS verarbeiteter Operator
op	logischer Operator
q	logische Anfrage
$\mathbf{S}$	Menge aller logischen Quelloperatoren
$\mathbf{z}$	logischer Operator - Senke
N	Anzahl Knoten in einem WSN
$\mathcal{N}$	vom INQP verarbeiteter Operator
$\bar{\mathbf{O}}$	Menge aller physischen Operatoren
$\bar{op}$	physischer Operator
$\bar{\mathbf{Q}}$	Menge aller äquivalenten physischen Ausführungsplan
$\bar{q}$	physischer Ausführungplan
$\bar{\mathbf{S}}$	Menge aller physischer Quelloperatoren
$\bar{\mathbf{z}}$	Menge aller physischen Senken
$\bar{\mathbf{z}}$	physischer Operator - Senke
Δt	Zeitintervall
f	Mittlerer Fanout eines Knotens
h	Höhe des WSN
i, j, k	Indexe
n	ein Sensorknoten
t	Zeitpunkt t
w	Fenstergröße
DSMS	*Data Stream Management System*
INQP	*In-Network Query Processor*
MOP	multikriterielles Optimum
WSN	*Wireless Sensor Network*

Kapitel 1

Einleitung

Im folgenden Abschnitt soll ein Überblick über das der Arbeit zugrunde liegende Thema gegeben werden. Nach einer generellen Einleitung in Abschnitt 1.1 werden im folgenden Abschnitt 1.2 verschiedene Szenarien als Motivation geliefert. Basierend auf diesen werden in Abschnitt 1.3 offene Herausforderungen abgeleitet, welche den wesentlichen wissenschaftlichen Beitrag der vorliegenden Arbeit beschreiben.

1.1 Motivation

In den letzten Jahren rückte die Verknüpfung von IT und realer Welt zunehmend in den Fokus von Wissenschaft und Industrie. Als Brücke dienen hierbei Aktoren bzw. Sensoren. Aktoren erlauben den gezielten Eingriff in die reale Welt und Sensoren erfassen Realwelt-Informationen und stellen diese in Form von analogen bzw. digitalen Signalen den informationsverarbeitenden Systemen zur Verfügung.

Mit fortschreitender Miniaturisierung im Bereich der mobilen Sensortechnik werden drahtlose Sensornetzwerke (*wireless sensor networks* - WSN) zunehmend interessanter für Anwendungen. Bei drahtlosen Sensornetzwerken handelt es sich üblicherweise um selbstorganisierende Verbunde von Sensorknoten. Jeder dieser Knoten ist für sich genommen ein Computer mit stark beschränkten Ressourcen. Im Regelfall verfügen diese über eine Sensorik zur Erfassung von Realweltdaten (zum Beispiel zum Messen von Luftfeuchtigkeit, Temperatur etc.), einer leistungsschwachen CPU, einer oftmals stark beschränkten Speichermenge für Daten und Programme, einer Batterie zur Energieversorgung und einem drahtlosen, zumeist funkbasierten Kommunikationsmodul[1]. Der Begriff „leistungsschwach“ bezieht sich hierbei auf den Vergleich zu moderner Computerhardware. Trotz Größenordnungen von nur einem Millimeter [211] bis zu mehreren Zentimetern im Durchmesser bewegen sich aktuelle Sensorknoten auf dem Leistungsstand von PC-Hardware Anfang der 1990'er Jahre, wobei der Energieverbrauch im Ver-

[1] Es existieren auch Projekte, welche die Kommunikation bzw. das Aufladen der Energieversorgung mittels Lichtwellen untersuchen [5].

lauf der Miniaturisierung drastisch gesenkt werden konnte und heute nur noch wenige mW bzw. W beträgt. Mit dem Ziel der Lebenszeitmaximierung wird die Konfiguration der Knoten zumeist in Abhängigkeit vom Anwendungsszenario gewählt.

In der Regel finden WSNs dort Anwendung, wo kurzfristig Messungen getätigt werden müssen oder wo eine Anpassung der Infrastruktur (z.B. eine Erweiterung der Energieversorgung) nicht möglich oder gewünscht ist. Typische Anwendungsgebiete sind zum Beispiel die Überwachung von Naturgebieten [41, 104, 150], die Verkehrs- oder die Gebäudeüberwachung [56, 62].

Bei der Entwicklung von Sensornetzwerken stellt die begrenzte Energieversorgung eine der größten Herausforderungen dar. Neben einer Vergrößerung des Energiespeichers führt insbesondere der effiziente Umgang mit den vorhandenen Energie-Ressourcen zu einer Erhöhung der zu erwartenden Lebenszeit. Hierbei bieten sich zwei generelle Ansätze an: (i) die Entwicklung besonders effizienter Hardware und (ii) der Einsatz besonders effizienter Algorithmen, Datenstrukturen und Kommunikationsprotokolle. Trotz der enormen Fortschritte im Bereich der Hardwareentwicklung in den letzten Jahren gibt es verschiedene Probleme, welche bis heute nicht behoben werden konnten. So ist zum Beispiel das Versenden von Daten unter Verwendung des Funkmodules um ein Vielfaches teurer als die Verarbeitung von Daten unmittelbar auf den Knoten unter Verwendung der zur Verfügung stehenden Ressourcen. Vor dem Hintergrund der Kommunikationsreduktion führen bestehende Applikationen für WSNs oftmals bereits eine (im Umfang stark beschränkte) Vorverarbeitung der gemessenen Daten auf den Sensorknoten aus. Lässt sich der Funktionsumfang dieser Applikationen dynamisch zur Laufzeit anpassen, so spricht man auch von *In-Network Query Processoren* (INQP).

Die von den Sensoren erfassten Daten werden zumeist an eine Basisstation gesendet, welche die empfangenen Daten auswertet bzw. sie an entsprechende Analyse-Systeme weiterleitet. Der von den Sensoren erzeugte Datenstrom (*data stream*) ist potentiell unendlich und kann in seiner Ankunftsrate stark variieren (zum Beispiel in Abhängigkeit der Abtast- bzw. Samplingrate der einzelnen Sensorknoten). Traditionelle Datenbank-Managementsysteme (DBMS) stoßen bei der Verarbeitung solcher Datenströme schnell an ihre Grenzen. So ist zum Beispiel das Speichern aller Datensätze eines Datenstromes und die anschließende Verarbeitung oftmals nicht möglich bzw. nicht erwünscht.

Datenströme werden fälschlicherweise oftmals mit Zeitreihen gleichgesetzt. Bei Zeitreihen handelt es sich zwar ebenso wie bei Datenströmen um Datenpunkte mit einem zeitlichen Bezug. Allerdings geht man bei der Zeitreihenanalyse davon aus, dass die vollständige Zeitreihe zum Analysezeitpunkt vorliegt (zum Beispiel in Datenbanken). Im Unterschied zur Datenstromanalyse ist somit ein wiederholter und insbesondere ein wahlfreier Zugriff auf die verschiedenen Datenpunkte in der Zeitreihe möglich.

Im Kontext von Datenströmen entwickelte sich eine neue Klasse von datenverarbeitenden Systemen, die *Datenstrom-Managementsysteme* (DSMS). Ursprünglich von den DBMS abgeleitet (STREAM [14]), wurden diese an die speziellen Herausforderungen von Datenströmen angepasst. Die wesentliche Innovation ist dabei die Verwendung von so genannten *one-pass*-Verfahren, bei denen der Algorithmus eingehende Werte nur

einmal betrachten kann. Anschließend werden die gelesenen Daten entweder verworfen oder in Form von Datenzusammenfassungen (Approximationen) temporär archiviert.

Zu den weiteren Herausforderungen von DSMS zählen der Umgang mit variablen Ankunftsraten und Echtzeitanforderungen bei der Verarbeitung von eingehenden Datensätzen. Um hierbei Warteschlangen und Blockierungen innerhalb des Systems zu vermeiden, verwerfen DSMS gezielt Datensätze. Das damit verbundene unvollständige Betrachten des Datenstromes führt zu approximierten Ergebnissen, deren Güte den vom Anwender definierten Mindestanforderungen genügen müssen.

Die Anfrageverarbeitung in einem DSMS unterscheidet sich von der eines DBMS grundlegend. In einem DBMS werden Anfragen initial einmal analysiert, optimiert und anschließend auf den aktuellen Datenbestand angewendet, d.h. die benötigten Daten werden einmal vom externen Speicher gelesen und von den einzelnen Operatoren bearbeitet. Wurde der letzte Datensatz gelesen, ist die Anfragebearbeitung beendet. Sämtliche für die Anfragoptimierung notwendigen Statistiken sind bereits zum Zeitpunkt der Optimierung bekannt. Im Gegensatz dazu endet die Anfragebearbeitung bei einem DSMS nicht, solange der Anwender diese nicht explizit stoppt. Da aber zum Zeitpunkt der Anfrageinitialisierung lediglich Statistiken über bisher gelesene Datensätze existieren, ist eine Optimierung unter Einbeziehung zukünftigen Verhaltens nur beschränkt möglich. Um dennoch eine effiziente Anfragebearbeitung gewährleisten zu können, sind das kontinuierliche Sammeln von Statistiken und die Reoptimierung von Anfragen zur Laufzeit (online-Optimierung) notwendig.

Neben der Beantwortung einfacher Anfragen ist die Extraktion von Mustern oder Regeln aus den zur Verfügung stehenden Daten eine der wesentlichen Aufgaben der Datenanalyse in DSMS. Die Wissensgewinnung (*knowledge discovery*) auf Basis großer Datenmengen wird im Allgemeinen auch als *Data Mining* bezeichnet. Neben der einfachen statistischen Analyse (zum Beispiel *avg*, *min*, *max*) stehen hier insbesondere komplexe Methoden wie zum Beispiel die Clusteranalyse, das Frequent Pattern Mining oder die Klassifikation von Werten im Mittelpunkt. Aufgrund der u.U. riesigen Datenmengen ist eine effiziente Verarbeitung bereits bei der Analyse in einem DBMS sehr wichtig, in einem DSMS aber unumgänglich. Zu den wesentlichen Herausforderungen bei der Adaption klassischer, aus dem Bereich der DBMS stammenden Data-Mining-Verfahren zählen die zu gewährleistende Online-Verarbeitung und der Umgang mit der Unvollständigkeit der Daten. Um den Mangel an echten Online-Verfahren auszugleichen, wird oftmals auch auf Batch-basierte Verfahren zurückgegriffen. Dabei werden Datenpakete (Batches) angesammelt, welche anschließend zusammen verarbeitet werden können.

1.2 Anwendungsszenarien

Eine sensorgestützte Überwachung findet man bereits heute in einer Vielzahl von Bereichen. WSNs hingegen findet man nur vereinzelt und dann zumeist in kleinen Szenarien. Im folgenden Abschnitt werden verschiedene Möglichkeiten des praktischen Einsatzes von WSNs aufgezeigt. Anhand dieser Beispiele lassen sich erste bis heute nicht gelöste

Herausforderungen ableiten, welche der folgenden Arbeit als Motivation zugrunde liegen.

Umweltüberwachung Ursprünglich zu militärischen Zwecken entwickelt, finden Sensornetzwerke heute oftmals im Bereich der Überwachung von Wäldern, Flüssen oder Seen Anwendung. Sie dienen dabei der Erkennung von Feuern, Überschwemmungen oder der Überwachung der Bewegung von Tiergruppen innerhalb einer Region. Aufgrund des Fehlens jeglicher Infrastruktur sind WSNs hierfür besonders geeignet.

Die Umsetzung der Überwachungslösung findet dabei im Wesentlichen in zwei Schritten statt: Zunächst wird vom Anwender definiert, welche Aufgabe das WSN zu erfüllen hat. Nachdem die Aufgabenstellung definiert wurde, muss deren Lösung in Software realisiert werden, welche anschließend auf die einzelnen Sensorknoten in der zu observierenden Region auszubringen ist. Im Allgemeinen verbleiben die Sensoren dann so lange im Überwachungsgebiet, bis sie ihre Aufgabe erledigt haben bzw. bis sie der Aufgabe auf Grund von zu geringen Energiereserven nicht mehr nachgehen können. Bei vielen der aktuellen Systeme ändert sich diese Aufgabe über die gesamte Zeit der Überwachung nicht.

Die Definition der Aufgabe und deren effiziente Lösung stellt aus softwaretechnischer Sicht die größte Herausforderung dar. Für einen möglichst langen Überwachungszyklus ist neben einer klar definierten Zielstellung gegenwärtig häufig auch Expertenwissen zur Entwicklung spezieller verteilter Algorithmen für WSNs notwendig. Dieses wiederum kann bei einer Vielzahl der Anwender nicht vorausgesetzt werden. Idealerweise sollte ein Anwender lediglich seine Ziele in einer einfachen Sprache definieren. Wie diese zu erreichen sind, d.h. welche speziellen Algorithmen eingesetzt werden und wie diese verteilt werden, sollte nicht durch den Anwender bestimmt werden. Idealerweise sollte das verwendete System die Aufgabe selbstständig analysieren und vollautomatisch die Softwareentwicklung vornehmen.

Gebäudeüberwachung Mit zunehmender Komplexität moderner Gebäude steigt die Nachfrage nach automatisierten Systemen, welche die Gesamtheit eines Gebäudes überwachen und Funktionsabläufe (zum Beispiel Regelkreisläufe für Heizung und Wasser) selbstständig steuern. Aufgrund ihrer Größe bieten moderne Büro- oder Geschäftsgebäude Raum für eine Vielzahl von Sensoren, Aktoren und Bedienelementen, welche miteinander vernetzt sind und automatisch die Regelung des Gebäudes übernehmen sollen. Im Mittelpunkt sich selbst überwachender Gebäude steht dabei zumeist eine Prozessoptimierung mit dem Ziel der Energie- und Kostenreduktion. Ein Beispiel aus diesem Bereich ist das Projekt *Customer Bautronic System*[2] (CBS). Bei diesem wird versucht, insbesondere den Nutzer in den Mittelpunkt der Prozessoptimierung zu stellen, so dass ein Gleichgewicht zwischen den durch das Gebäude verursachten Kosten und der Zufriedenheit des Nutzer entsteht.

[2] /www.customerbautronic.de

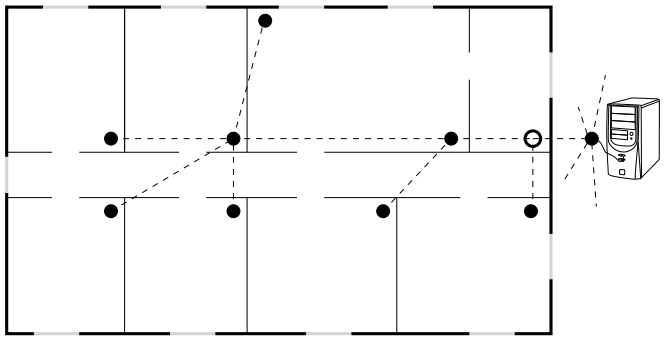


Abbildung 1.1: Szenario Gebäudeüberwachung

Ein wichtiges Kriterium für die optimale Steuerung eines Gebäudes ist dabei der intelligente Einsatz von Sensoren. Mit einer zunehmender Anzahl von Sensoren steigt auch die zu verarbeitende Datenmenge und letztendlich der Aufwand, welcher zur Extraktion von Wissen aus den damit verbundenen Datenströmen notwendig ist. Um sowohl den Aufwand für die kontinuierliche Überwachung als auch die Kosten für die Anschaffung von Sensorhardware möglichst gering zu halten, ist aus den möglichen Messpunkten in einem Gebäude die Menge zu extrahieren, welche für eine spätere Analyse ausreichend genaue Daten liefert und den Aufwand an eingesetzter Hardware minimiert. Da Berechnungen während der Entwurfsphase eines Gebäudes lediglich erste Hinweise auf mögliche Einsatzorte liefern, sind zusätzlich reale Messungen an einer Vielzahl von Messpunkten während der Bauphase notwendig. Für diese initiale Überwachungsphase, welche sich zwischen wenigen Tagen und mehreren Wochen bewegen kann, sind WSNs ein möglicher Lösungsweg. Aufgrund der frühen Phase im Bauabschnitt und der hohen Anzahl von Messpunkten kann dabei eine externe Versorgung der Sensoren mit Energie nicht immer gewährleistet werden. Der verhältnismäßig lange Messzeitraum mit unter Umständen höheren Samplingraten, als dies im späteren Betrieb notwendig ist (der optimale Samplingzeitpunkt muss erst noch ermittelt werden), macht einen effizienten Umgang mit der zur Verfügung stehenden Energie unumgänglich.

Wurde die initiale Überwachung des Gebäudes beendet, können die Sensoren des WSN wieder aus dem Gebäude entfernt und durch die für die dauerhafte Messung vorgesehenen Sensoren ersetzt werden. Typischerweise ändern sich die Aufgaben, welche die Sensoren während der Testphase bearbeiten, nicht, so dass zur Laufzeit auf die Software der Knoten kein (oder nur ein sehr geringer) Einfluss genommen wird. Werden die Sensoren für die Überwachung eines weiteren Gebäudes konfiguriert, können eventuell Änderungen an der Software notwendig werden.

Bei drahtlosen Sensornetzwerken handelt es sich üblicherweise um selbstorganisierende Ad-Hoc-Netzwerke, d.h. für den Multihop-Datentransfer von einem Knoten zu einem anderen bauen die Sensorknoten selbstständig eine Pfadstruktur auf. Diese Routingstruktur ist dabei abhängig von den Eigenschaften der Knoten (zum Beispiel der Sendeleistung und dem Batteriestatus), deren Lage zueinander und wird dynamisch zur Laufzeit angepasst. Ausgefallene Knoten können zur Laufzeit dynamisch kompen-

siert werden. Zudem wird versucht, für den Datentransfer anfallende Energiekosten möglichst über alle Knoten gleichmäßig zu verteilen. Zusätzlich zu diesen sich selbst aufbauenden Routingstrukturen kann eine logische Netzwerkschicht sinnvoll sein. So verfügt der Nutzer unter Umständen über Wissen, welches für eine effizientere In-Network-Bearbeitung von Nutzen sein kann. Im Fall der Gebäudeüberwachung können zum Beispiel Gebäudestrukturen (Räume, Etagen, Flügel usw.) direkt in den Verarbeitungsprozess integriert werden. So ist es möglich, dass ein einzelner leistungstärkerer Sensorknoten für die Auswertung der Daten einer Etage verantwortlich ist. Die (aggregierten) Ergebnisse sendet dieser dann an einen Knoten, welcher für den Gebäudeflügel zuständig ist (siehe Abbildung 1.1).

Patientenüberwachung Die zunehmende Miniaturisierung der Sensorknotenhardware eröffnet eine Vielzahl neuer Anwendungen. So lassen sich zum Beispiel Kleinstsensoren bereits heute in Freizeitbekleidung integrieren und erlauben ihrem Träger eine kontinuierliche Überwachung seiner Vitalfunktionen. In zukünftigen Applikationen könnten diese zudem via drahtloser Kommunikation (zum Beispiel über das Mobiltelefon des Trägers) direkt an eine Basisstation oder im Notfall an einen Arzt gesendet werden.

Im medizinischen Umfeld eröffnet der Einsatz von WSNs interessante Möglichkeiten. So können zum Beispiel Patienten in Krankenhäusern auch außerhalb ihrer Betten überwacht werden, ohne Beeinträchtigungen durch störende Sensorik hinnehmen zu müssen. Die Überwachung von Patienten (*personal health monitoring*) ist allerdings auch mit neuen Herausforderungen verbunden [161,205]. So gilt es, neben dem Problem der sozialen Akzeptanz auch Probleme technischer Natur und Sicherheitsaspekte zu lösen.

Im Fall des personal health monitoring ist jeder Patient mit einer Menge an physischen Sensoren ausgestattet (zum Beispiel zum Erfassen der Herzfrequenz, des Pulses oder der Sauerstoffkonzentration im Blut). Jeder dieser Sensoren sendet seine Messwerte (kabellos oder kabelgebunden) an einen körpernahen Empfänger. Der Empfänger verarbeitet die von den Sensoren gelieferten Daten (er reduziert die zu sendende Datenmenge) und überträgt diese an einen zentralen Server, welcher die Patientendaten analysiert und für eine Auswertung durch den Arzt vorhält.

Ist eine Person Träger der Sensorik, ergeben sich eine Vielzahl neuer Probleme. Neben der notwendigen Akzeptanz von Seiten des Nutzers und der kontinuierlichen Überwachung ist insbesondere ein hoher Tragekomfort wichtig. Nur wenn die Sensorik den Patienten nicht stört, wird diese auch von ihm akzeptiert. Daher müssen am Körper getragene Sensoren möglichst klein und unauffällig sein. Dies resultiert allerdings u.a. in kleineren Energiespeichern, was wiederum die Lebensdauer der Sensoren verringert. Zudem begrenzen die kompakten Maße die Möglichkeiten für den weiteren Umgang mit den Messwerten. In kleineren Geräten kommt entsprechend leistungsschwächere Hardware zum Einsatz. Dennoch sollten solche Geräte möglichst universell einsetzbar sein. Ein Patient, welcher auf verschiedene physiologische Werte hin überwacht werden

muss, sollte nach Möglichkeit nicht für jeden der zu messenden Werte ein komplettes System bei sich tragen müssen.

Der Einsatz verschiedener Systeme würde auch einen hohen Kostenfaktor für das Krankenhaus mit sich bringen. Idealerweise werden lediglich notwendige physischen Sensoren an den lokalen Empfänger des Patienten angeschlossen. Der Empfänger übernimmt anschließend die Weiterverarbeitung der erfassten Werte und die notwendige Datenübertragung. Eines der dabei auftretenden Probleme ist die Komplexität bzw. die Vielfalt der für die Verarbeitung auf dem Empfänger notwendigen Software. Unterschiedliche Patienten bzw. Krankheitsbilder erfordern üblicherweise auch eine unterschiedliche Auswertung der erfassten Werte. Da die für die Messung und Verarbeitung notwendige Hardware zur Vermeidung zusätzlicher Energiekosten zumeist auf das Wesentliche reduziert ist, lässt sich oftmals nur ein stark beschränkter Funktionsumfang in die Sensorapplikationen integrieren. Wechselt der Träger der Sensorik, dann muss u.U. eine Anpassung auf der Softwareseite vorgenommen werden. Sind die vom Arzt gewünschten Analysen zu komplex für eine vollständige Verarbeitung auf den Knoten bzw. verfügen die mobilen Geräte nicht über ausreichend Ressourcen, dann muss die Auswertung teilweise auf der zentralen Instanz stattfinden. Die Entscheidung, inwieweit Anwendungen von einer zentralen Instanz bzw. von den Sensorknoten ausgeführt werden, sollte nicht durch den Nutzer (in diesem Fall den Arzt oder das betreuende Personal) getroffen werden. Der Entwurf entsprechender Systeme ist zeitaufwendig und erfordert vom Nutzer zudem ein erhebliches Hintergrundwissen, welches nicht in jedem Fall vorausgesetzt werden kann.

Eine weitere Herausforderung im Kontext von personal health monitoring ist die u.U. enorm hohe Menge an Daten, welche von den Sensoren produziert und auf der zentralen Instanz verarbeitet und zwischengespeichert werden muss. In einem Großkrankenhaus gibt es oft Hunderte von Patienten, welche oftmals mit einer Vielzahl von physischen Sensoren zur Überwachung ausgestattet sind. Mit unterschiedlicher Auslastung des Krankenhauses kann es hierbei zu enormen Schwankungen im auftretenden Datenvolumen kommen. Der effiziente Umgang mit den zur Verfügung stehenden Ressourcen ist hier ein zentrales Problem. Die einfachste Lösung wäre das Abfangen von Datenspitzen durch den Ausbau der Infrastruktur, was wiederum mit einem erheblichen Kostenaufwand verbunden ist. Alternativ könnte sich das System dynamisch an das schwankende Datenaufkommen anpassen. Auf Hardwareebene bedeutet dies das dynamische Hinzufügen bzw. Freigeben von Ressourcen (zum Beispiel die Auslagerung der Verarbeitung in ein entsprechendes Cloud-System[3]). Im Bereich der Applikationen sind eventuelle Anpassungen der Algorithmen so wie deren Parameter oder Datenstrukturen möglich.

[3]Unter Cloud-Computing versteht man Systeme, bei denen Ressourcen und Anwendungen über Netzwerk zur Verfügung stehen und die sich dynamisch an die jeweiligen Anforderungen anpassen.

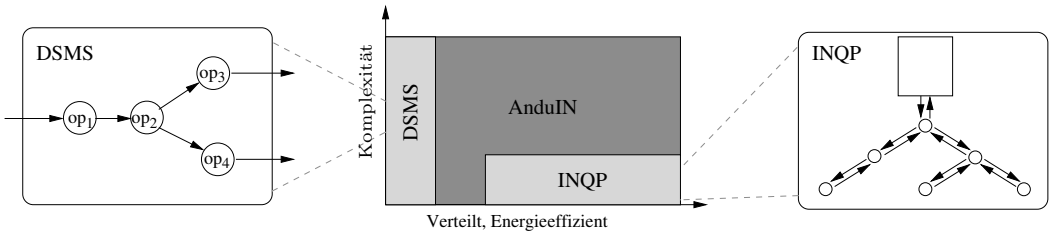


Abbildung 1.2: Ziel dieser Arbeit

1.3 Ziele dieser Arbeit

Im letzten Abschnitt wurden mögliche Anwendungsszenarien für drahtlose Sensornetzwerke vorgestellt. In den letzten Jahren entstand bereits eine Reihe von Systemen, welche die in den Szenarien auftauchenden Probleme teilweise lösen. Dabei beschränken sich diese zumeist auf die Lösung einiger Teilprobleme. Im Wesentlichen existieren zwei Klassen von Systemen für die Verarbeitung von kontinuierlich erzeugten Sensordaten: INQP und DSMS. Aktuelle INQP erlauben Ad-Hoc-Anfragen an das WSN mit dem Ziel einer möglichst energieeffizienten Beantwortung. Aufgrund des geringen Speichers der Sensorknoten fällt deren Operationsumfang sehr gering aus und beschränkt sich zumeist auf primitive Operationen wie Filter, Verbund oder Aggregation.

Im Gegensatz dazu stellen aktuelle DSMS neben den primitiven Operationen zumeist auch eine Vielzahl an komplexen Data-Mining-Operationen zur Verfügung. Eine einfache Möglichkeit, Datenströme aus einem WSN einer komplexen Verarbeitung zukommen zu lassen, ist die Erfassung der Daten durch den INQP, das Senden an die zentrale Instanz und die anschließende Weiterverarbeitung durch ein DSMS. Problematisch ist an dieser Stelle das kostenintensive Versenden der Rohdaten an das DSMS.

Im Mittelpunkt dieser Arbeit steht die Entwicklung eines kombinierten DSMS/INQP-Systems, welches die vollständige bzw. teilweise Verarbeitung von nahezu beliebig komplexen Operatoren in drahtlosen Sensornetzwerken unterstützt (siehe Abbildung 1.2). Die Entscheidung, welche Teile einer Anwendung in das WSN ausgelagert werden, soll dabei autonom durch das System auf Basis nutzerspezifizierter Eigenschaften geschehen. Definierte Anfragen sollen hierzu vom System automatisch analysiert, evaluiert und für die getrennte Verarbeitung in der DSMS-Komponente und im INQP zerlegt werden. Dem Nutzer soll ein entsprechendes Interface bereitgestellt werden, welches ihm die deklarative Beschreibung seiner Ziele ermöglicht. Wie diese Ziele erreicht werden, wird durch das System bestimmt.

Weiterhin sollen vom entwickelten System die folgenden Anforderungen betrachtet werden:

- *Manuelles Netzwerk-Deployment*: Der Nutzer soll in die Lage versetzt werden, manuell logische Netzwerktopologien zu definieren. Diese sollen eine Optimierung von Routing- und Verarbeitungsprozessen erlauben.

- *Multikriterielle Optimierung*: Wie bereits erwähnt ist das primäre Optimierungsziel des zu entwickelnden Systems die Minimierung des Energieverbrauches der Sensorknoten. Aus den oben beschriebenen Szenarien wird ersichtlich, dass es weitere mögliche Optimierungskriterien gibt. Im Beispiel der Patientenüberwachung stellt das Abfangen von Lastspitzen auf der zentralen Instanz ein weiteres Optimierungsziel dar. Eine der Anforderungen an das zu entwickelnde System ist somit die Möglichkeit einer Optimierung hinsichtlich mehrerer, möglicherweise konkurierender Ziele.

- *Dynamische Anpassung von Anfragen*: Aufgrund der sich u.U. ändernden Charakteristika der erfassten Messdaten ist eine kontinuierliche Überwachung und gegebenenfalls notwendige Adaption von laufenden Anfragen notwendig. Bei der Reorganisation muss dabei auf die Besonderheiten der beiden jeweiligen Systemkomponenten INQP und DSMS eingegangen werden.

Das zu entwickelnde System soll prinzipiell ein Framework für die einfache Integration weiterer Techniken zur Datenanalyse bereitstellen. Neben den klassischen Verfahren zur Auswertung einfacher Datenströme erfordern Sensornetzwerke spezielle Lösungen für den Umgang mit den im Netzwerk erzeugten Daten. Zusätzlich bieten sie aber auch Potential für die Integration neuer interessanter Analyseverfahren. Aus diesem Grund sollen neben den klassischen Verfahren aus dem Bereich der Datenstromverarbeitung zusätzlich die folgenden Problemstellungen vom zu entwickelnden System betrachtet werden:

- *Analyse über räumlichen Daten:* Bei Sensornetzwerken handelt es sich immer um räumlich verteilte Systeme. Oftmals ist der Anwender aber nicht an einer Auswertung von Daten aus dem vollständigen System interessiert. Verfügen die Sensoren über Techniken zur Positionsbestimmung bzw. ist eine Zuordnung zwischen Sensoren und deren Lokation eindeutig möglich, dann sollen diese durch das System ausgewertet werden können. Hierzu sollen entsprechende Verfahren zur Analyse von Datenströmen mit räumlichem Bezug in das zu entwickelnde System integriert werden.

- *Data Cleaning:* Die Verarbeitung von Daten aus WSNs stellt Anwender und Anwendung häufig vor spezielle Probleme, welche in klassischen DBMS nicht auftreten. So können zum Beispiel Probleme beim Erzeugen oder beim Transfer der Daten zum Empfänger Einfluss auf die Datenqualität und somit auch auf die Güte von Zwischen- bzw. Endergebnissen haben. Die Datenaufbereitung (*data cleaning*) gehört daher zu einer der grundlegenden Aufgaben eines datenstromverarbeitenden Systems, welches auf Sensordaten operiert. Im Verlauf der vorliegenden Arbeit soll hierzu ein spezielles Verfahren zur Erkennung von anormalem Verhalten in trend-behafteten Daten in Datenströmen entwickelt werden.

- *Data Mining:* Wie bereits erwähnt, soll das zu entwickelnde System die Integration komplexer Verfahren in den Analyseprozess ermöglichen. Stellvertretend für diese Klasse von Verfahren soll ein klassisches Data-Mining-Verfahren integriert werden.

In den letzten Jahren wurde bereits eine Vielzahl der bekannten Data-Mining-Verfahren in die Welt des Data Stream Mining transferiert. Die Verarbeitung von Sensordaten stellt hierbei jedoch eine besondere Herausforderung dar. Grundlegende Data-Mining-Verfahren wie zum Beispiel die Klassifikation oder das Frequent Pattern Mining setzen dabei zumeist kategorische Attribute voraus. Bei den von Sensorknoten bereitgestellten Daten handelt es sich allerdings in den allermeisten Fällen um Attribute über einem kontinuierlichen Wertebereich (quantitative Attribute). Im Bereich der Klassifikation existieren bereits entsprechende partitionierende Lösungen für diese Art von Attributen. Für das Problem des Frequent Pattern Mining über quantitativen Attributen sind gegenwärtig lediglich Lösungen für statische Daten bekannt. Ein entsprechendes Verfahren, welches für die Analyse von Datenströmen geeignet ist, soll daher im Laufe der Arbeit beschrieben und in das zu entwickelnde System integriert werden.

1.4 Gliederung

Die Arbeit ist im Weiteren wie folgt gegliedert: Zunächst folgt in Kapitel 2 ein Überblick über verwandte Arbeiten. Basierend auf dieser Übersicht werden offene Probleme abgeleitet, welche durch das im Rahmen dieser Arbeit entwickelte System gelöst werden sollen.

Kapitel 3 widmet sich den prinzipiellen Anforderungen an das zu entwickelnde System und bietet einen generellen Überblick über dessen einzelne Komponenten.

Kapitel 4 betrachtet die Anfrageverarbeitung im entwickelten System im Detail. Ein Schwerpunkt hierbei ist das verwendete Kostenmodell, welches die Grundlage für die automatische Auswahl des optimalen Anfrageplanes bildet. Weiterhin wird in Kapitel 4 auf das Problem der adaptiven Anfrageoptimierung und die Multi-Anfrage-Optimierung eingegangen.

Nachdem in den vorangegangenen Kapiteln das System und dessen Arbeitsweisen vorgestellt wurden, folgt in Kapitel 5 eine Übersicht der in das System integrierten Operatoren. Dies beinhaltet eine detaillierte Beschreibung der Integration von Operatoren mit räumlichem Bezug (Abschnitt 5.1) sowie ein Verfahren zur adaptiven Burst-Erkennung (Abschnitt 5.2) und ein Verfahren zum Frequent Pattern Mining quantitativer Attribute (Abschnitt 5.3).

Kapitel 6 beschreibt Implementierungsdetails des entwickelten Prototyps und präsentiert Evaluierungsergebnisse, welche auf Basis des Prototyps gewonnen werden konnten.

In Kapitel 7 werden eine Zusammenfassung der präsentierten Arbeit und ein Ausblick auf mögliche zukünftige Tätigkeiten in diesem Umfeld gegeben.

Kapitel 2

Verwandte Arbeiten

In der Literatur finden sich zahlreiche Arbeiten, welche sich mit dem Problem der Datenverarbeitung innerhalb von Sensornetzwerken (*In-Network*) bzw. einer generellen Datenstromverarbeitung auseinandersetzen. Zumeist erfolgten die Betrachtungen dabei unabhängig voneinander. Vereinzelte Ansätze versuchen, paradigmenübergreifende Lösungen herzustellen. Das nachstehende Kapitel soll einen generellen Überblick über relevante Arbeiten der letzten Jahre in beiden Forschungsbereichen geben (siehe Abbildung 2.1). Details zu speziellen Algorithmen werden in den weiteren Kapiteln an passender Stelle ausgeführt.

Zunächst wird in Abschnitt 2.1 ein Überblick über aktuelle Entwicklungen im Bereich der drahtlosen Sensornetzwerke gegeben. Da bei einem WSN alle Komponenten in die Energiebilanz eingehen, soll zunächst ein Überblick über die wesentlichen Teile (Betriebssysteme, Routing-/Aggregationsprotokolle, In-Network Query-Prozessoren) und deren Funktionsweise gegeben werden.

In Abschnitt 2.2 wird ein Überblick über Arbeiten im Bereich der Datenstromverarbeitung gegeben. Dabei wird insbesondere auf die Probleme bei der Entwicklung von Algorithmen für das Data Mining, bestehende Stream-Mining-Lösungen und die Opti-

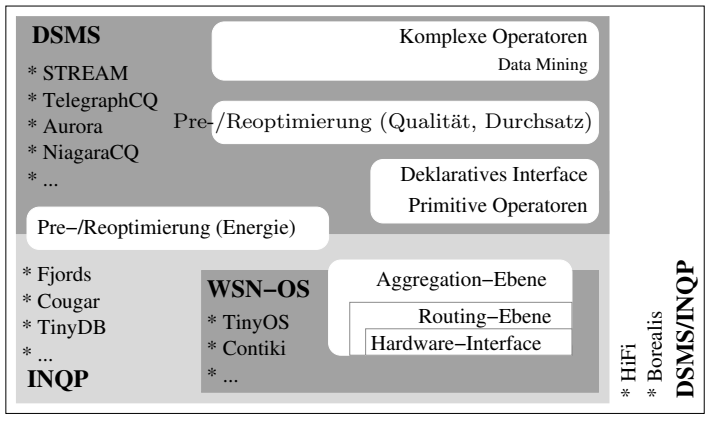


Abbildung 2.1: Überblick über existierende DSMS/INQP-Lösungen

mierung von kontinuierlichen Anfragen eingegangen

Ein Schwerpunkt dieser Arbeit ist die teilweise bzw. vollständige Verarbeitung komplexer Prozesse innerhalb des WSN. Abschnitt 2.3 liefert einen kurzen Überblick über bekannte Data-Mining-Verfahren sowie eventuell existierende In-Network-Lösungen.

In Abschnitt 2.4 werden die vorgestellten Verfahren analysiert und offene Herausforderungen herausgestellt.

2.1 Drahtlose Sensornetzwerke

Mit der zunehmenden Miniaturisierung der Sensorhardware wurde auch die Entwicklung von neuen Anwendungen beschleunigt. Betrachtet man Software für Sensorknoten, so ist es nur schwer möglich, einzelne Komponenten oder Techniken allein zu betrachten. Einen sehr guten Überblick über aktuelle Konzepte und Techniken im Bereich der drahtlosen Sensornetzwerke geben die Autoren in [165]. Im Folgenden soll ein grundlegender Überblick über aktuell bestehende Systeme und Techniken gegeben werden.

2.1.1 Betriebssysteme

Die geringen Hardwareressourcen der einzelnen Sensorknoten führten zur Entwicklung spezieller an die Anforderung von WSN angepasster Betriebssysteme. Der Umfang an möglichen Betriebssystemen für Sensorknoten ist ebenso vielfältig wie die Anzahl an Hardwareplattformen. Die Systeme reichen dabei von primitiven Lösungen, welche lediglich die Hardwareansteuerung übernehmen und eine einfache P2P Kommunikation erlauben, bis hin zu Lösungen, welche sowohl Multi-Threading bzw. sogar die Ausführung einer einfachen Java-Virtual-Maschine ermöglichen (SOS [100], Mantis [33], t-kernel [92]).

Eines der am weitesten verbreiteten Sensorbetriebssystem ist TinyOS [58,106]. Anfänglich wurde es an der Berkeley Universität entwickelt und ging später in ein Open-Source-Projekt mit zeitweise mehreren tausend Entwicklern über. TinyOS ist ein ereignisbasiertes Betriebssystem, d.h. für anstehende Aufgaben verlässt der Sensorknoten den energiesparenden *Sleep Mode*, führt das Ereignis schnellstmöglich aus und versetzt sich anschließend wieder in den Ruhezustand.

TinyOS steht in Form einer Bibliothek zur Verfügung. Je nach Anwendung werden die entsprechenden Komponenten mit dem Betriebssystemkern verknüpft und anschließend auf die Sensorknoten verteilt. TinyOS erlaubt das Verteilen von Updates, sowohl kabelgebunden (UISP-Schnittstelle) als auch über die in den Knoten integrierte Funkhardware (*Over-the-air-programming* (OTA)). Aufgrund der modularen Bauweise ist es auf einfache Weise möglich, TinyOS für spezielle Applikationen und verschiedene Hardwareplattformen anzupassen.

Ursprünglich wurde TinyOS in der Programmiersprache *C* implementiert. Aufgrund der Komplexität von C und dessen Bibliotheken wurde mit *nesC* eine speziell an die

Anforderungen von Sensorknoten angepasste Programmiersprache entwickelt. Zu den wesentlichen Nachteilen von TinyOS gehören das Fehlen echter Threads sowie von virtuellem und dynamischem Speicher. Des Weiteren ist es nicht möglich, Aufgaben zu priorisieren.

Das für die Evaluierung dieser Arbeit eingesetzte Betriebssystem *Feuerware* der FU Berlin[1] basiert in wesentlichen Teilen auf dem Betriebssystem Contiki (*Operating System for Embedded Smart Objects*) [70]. Contiki wurde ursprünglich am Schwedischen Institut für Informatik entwickelt, wird mittlerweile aber auch in einer Vielzahl von Projekten und auf unterschiedlichster Hardware (8-Bit Systeme) eingesetzt. Der Einsatz von Contiki ist somit nicht nur auf Sensorknoten beschränkt.

Zu den wesentlichen Vorteilen von Contiki gegenüber TinyOS zählen die Unterstützung von ANSI *C99* und die Möglichkeit der Verwendung von Threads, den so genannten *Protothreads* [71]. Weiterhin werden von Contiki zusätzlich die Kommunikationsprotokolle μIP und Rime unterstützt, wobei insbesondere μIP bisher im Bereich der Sensorknoten einzigartig ist. Bei μIP handelt es sich um einen speziell auf die Hardwarevoraussetzungen im Bereich der Sensornetzwerke angepassten TCP/IP Protokollstapel. Prinzipiell können Sensorknoten damit als herkömmliche PC's in einem lokalen Netzwerk betrachtet werden. Insofern die Hardware es zulässt, ist somit auch die Nutzung entsprechender Netzwerkdienste wie zum Beispiel DHCP oder Apache möglich.

2.1.2 Routing-/Aggregationsprotokolle

Im Gegensatz zu klassischen Netzwerken, bei welchen zumeist das Finden des kürzesten Pfades zwischen zwei Knoten im Mittelpunkt steht (*address centric approach*), entwickelte man für drahtlose Sensornetzwerke Routingprotokolle, welche insbesondere die Aggregation von Daten in den Mittelpunk stellen. Man spricht in diesem Zusammenhang auch von datenzentriertem Routing bzw. einem datenzentrierten Ansatz (*data centric approach*). Statt den kürzesten Routing-Pfad zwischen zwei oder mehr Sensorknoten zu wählen, wird beim datenzentrierten Ansatz der Pfad gewählt, welcher das energieeffiziente Einsammeln von Daten zum Ziel hat. Somit kann ein Pfad zwar länger sein, insgesamt wird für die Übertragung der Daten allerdings weniger Energie benötigt.

Typischerweise besitzt jeder Sensorknoten eine kurze Liste mit Informationen über seine Nachbarn sowie deren Verbindungen zum Netzwerk. Um eine möglichst energieeffiziente Verarbeitung zu gewährleisten, entstand im Laufe der Zeit eine Vielzahl von Routingprotokollen. Im Wesentlichen lassen sich diese nach der ihnen zugrunde liegenden Struktur in die beiden Klassen: (i) flache Netzwerke und (ii) hierarchische Netzwerke unterteilen [176]. Zu den bekanntesten flachen Netzwerken zählen *push diffusion* [105] und *directed diffusion* [111]. Bei der *push diffusion* erfolgt die Verbreitung der Daten von der Quelle aus. Interessierte Senken müssen sich bei einer Quelle registrieren, um anschließend die Daten zu erhalten. Im Gegensatz dazu geht die Interaktion bei Verfahren der *directed diffusion* von den Quellen aus. Diese senden ihre Anfragen via Broadcast

[1] http://cst.mi.fu-berlin.de/projects/ScatterWeb/

an alle Knoten. Falls möglich, antwortet die betroffene Quelle, wobei die Nachrichten zunächst mit niedriger Datenrate auf verschiedenen Pfaden zur Quelle gesendet werden. Anschließend wählt die Quelle den erfolgversprechendsten Pfad aus. Weitere Nachrichten werden dann ausschließlich über diesen Pfad zur Senke gesendet.

Chained-based aggregation, *tree-based aggregation* und *cluster-based aggregation* zählen zu den hierarchischen Routingprotokollen. Bei der *chained-based aggregation* werden Nachrichten entlang einer Kette jeweils zwischen den beiden Knoten mit der geringsten Distanz ausgetauscht. Die Effizienz beim Versenden von Nachrichten entlang der verwendeten Kette ist dabei extrem stark von dem verwendeten Algorithmus zur Kettenbildung abhängig. Eines der größten Probleme des kettenbasierten Routings ist die geringe Robustheit gegen Knotenausfälle. Bei einem Knotenausfall muss eine komplette Reorganisation erfolgen, da ansonsten alle Knoten hinter dem ausgefallenen nicht mehr sichtbar sind. Außerdem setzen Protokolle wie zum Beispiel PEGASIS [144] globales Wissen zur optimalen Distanzbestimmung voraus.

Bei den *tree-based aggregation*-Netzwerken sind die Knoten in einer Baumstruktur organisiert, wobei die Datenaggregation von den inneren Knoten durchgeführt wird. Zu den Nachteilen der baum-basierten Protokolle zählen der hohe Konstruktionsaufwand sowie deren geringe Robustheit beim Ausfall von Knoten. Eines der bekanntesten baumbasierten Protokolle ist TAG (*Tiny AGgregation*) [147], welches als Aggregationsprotokoll für auf TinyOS basierende Sensornetzwerke entwickelt wurde. Bei TAG warten Elternknoten auf die Nachrichten ihrer Kindknoten, aggregieren diese und senden die Daten-Zusammenfassungen anschließend an ihre Elternknoten.

Bei der *cluster-based aggregation* wählen die Knoten eines Clusters, d.h. geografisch nah beieinander liegende Knoten, einen Repräsentanten (*cluster head*), welcher die Aggregation übernimmt und mit der Senke bzw. anderen Repräsentanten kommuniziert. Eines der interessantesten Probleme bei der *cluster-based aggregation* ist die geeignete Auswahl des Cluster-Repräsentanten, da dieser höheren Belastungen unterliegt als andere Knoten. Verfahren wie zum Beispiel LEACH [170] wechseln daher die Repräsentanten in Abhängigkeit verschiedener Kriterien dynamisch aus.

Wie eingangs bereits erwähnt, ist der Übergang bei drahtlosen Sensornetzwerken zwischen Betriebssystem und Anwendung fließend. So präsentierten Madden et al. in [149] im Kontext von TinyDB zum Beispiel den *Semantic Routing Tree* (SRT). Bei diesem ist die Routingstrategie Teil der Anwendung. Im Gegensatz zu den bisher präsentierten Lösungen berücksichtigt der SRT die Anfragesemantik. Mit dessen Hilfe können Knoten selbstständig entscheiden, ob Kindknoten zur Bearbeitung einer Anfrage beitragen können. Ein SRT kann als eine zusätzliche Netzwerkschicht betrachtet werden, welche als eine Art Index fungiert. Erhält ein Knoten eine Anfrage, so kann dieser prüfen, ob er Kindknoten kennt, welche zur Anfragebearbeitung beitragen können. Sollte dies der Fall sein, so wird die Anfrage an den entsprechenden Knoten weitergeleitet. Kennt er keinen solchen Knoten und kann selber auch nicht zur Anfragebearbeitung beitragen, so verwirft der Knoten die Anfrage.

Wie gezeigt wurde, existiert eine Vielzahl von Lösungen zum Finden des Routingpfades.

Bei allen publizierten Verfahren handelt es sich jedoch um Heuristiken, da das Problem das Finden des optimalen Pfades NP-Hart ist [135].

Die Autoren in [166] beschreiben ein System, welches die Definition logischer Nachbarschaftsbeziehung mittels der Anfragesprache *SPIDEY* ermöglicht. SPIDEY ist dabei als eine Erweiterung zu TinyOS gedacht, welche dessen physische Nachbarschaften auf logischer Ebene erweitert. Die Nachbarschaftsbeziehungen werden dabei über Templates erstellt. In diesen beschreibt der Nutzer, über welche Eigenschaften Knoten verfügen müssen, um sich zu einer Nachbarschaft zusammenzuschließen. Die Knoten prägen diese Templates dann zur Laufzeit aus.

2.1.3 In-Network Query-Prozessoren

Im Mittelpunkt der INQP-Entwicklung steht die effiziente teilweise oder komplette Anfrage-Verarbeitung durch das Sensornetzwerk.

Fjords Eines der ersten Systeme, welches die Besonderheiten von Sensorknoten berücksichtigte, war Fjords (*Framework in Java for Operators on Remote Data Streams*) [148]. Bei Fjords handelt es sich noch nicht um einen vollwertigen INQP. Die Sensoren führen lediglich Messungen durch und senden ihre Ergebnisse an einen Proxy-Server, welcher die Ergebnisse zwischenspeichert, eine einfache Datenvorverarbeitung vornimmt und die Kommunikation mit dem (Ziel-)Server regelt. Außerdem ist der Proxy-Server in der Lage, die Sampling-Raten der in seinem Verantwortungsbereich liegenden Sensoren dynamisch anzupassen. Eines der wesentlichen Ziele von Fjords war die Unterstützung von sowohl push-basierten als auch pull-basierten Anfragen. Da die Sensoren selber nur ein kontinuierliches Versenden von Ergebnissen erlaubten, erfolgte die Anfragebeantwortung auf Basis der auf den Proxy-Servern gecachten Daten.

Cougar Cougar [223, 224] war eines der ersten Projekte, welches das Konzept der verteilten Datenbank auf WSNs übertrug. Im Rahmen von Cougar wurde eine verteilte Architektur zur Durchführung von Aggregationen und komplexeren Algorithmen präsentiert. Kern der Architektur ist eine zusätzliche Anfrageschicht auf jedem Sensorknoten, welche die Schnittstelle zwischen Anwendungen und Sensornetzwerk bildet. Cougar unterscheidet drei Klassen von Knoten: Quellknoten, innere Knoten und Gate-Knoten. Die Quellknoten dienen dabei ausschließlich der Erfassung von Messwerten. Die inneren Knoten führen zudem noch einfache Berechnungen durch. Der Gateknoten dient u.a. als Verbindung zu Datensenken auf einer zentralen Instanz und verfügt zudem über einen einfachen Anfrageoptimierer. Im Kontext von Cougar werden auch erste Ansätze zur In-Network-Aggregation präsentiert [222]. Dies beinhaltet zum einen das einfache Zusammenfügen und Versenden von Nachrichten (*packet merging*) und zum anderen die partielle Berechnung von Aggregaten auf den inneren Knoten (*partial aggregation*).

TinyDB TinyDB [146, 149], ein auf TinyOS aufbauender INQP, ist der wohl bekannteste INQP. Die Entwickler bezeichnen TinyDB als ein deklaratives Datenbanksystem für drahtlose Sensornetzwerke. Alle von den Sensorknoten erzeugten Daten werden in einer logischen Tabelle (`sensor`) verwaltet, welche durch die im Kontext von TinyDB entwickelte Anfragesprache ACQL (*Acquisutional Query Language*) abgefragt werden kann. Knoten werden in `sensor` durch Zeilen repräsentiert. Spalten in `sensor` repräsentieren die Attribute der Knoten (z.B. Temperatur, Luftfeuchtigkeit etc.). Gemessene Werte werden dabei nur kurze Zeit (zum Zeitpunkt der Anfrage) in `sensor` vorgehalten.

Die wesentlichen, von TinyDB unterstützten Operatoren sind das Sampling (das Akquirieren von Daten), die Selektion, die Aggregation, die Gruppierung und der Join [146], wobei der Verbund als einfacher Nested Loop Join umgesetzt wurde.

Im Zusammenhang mit TinyDB wurde ACQL entwickelt. ACQL folgt dem Anfragekonstrukt `SELECT-FROM-WHERE-GROUPBY`, wobei es mit Erweiterungen für das Handling von Sensorknoten erweitert wurde. Zu den wichtigsten Erweiterungen zählen die Möglichkeit der Angabe von Sampling-Intervallen (`SAMPLE-PERIOD`-clause) und die Möglichkeit, Aggregation auch über Zeitfenster durchzuführen. Eine der wesentlichen Erweiterungen von ACQL ist die Einführung von *Events*. Sie erlauben es, Knoten nur beim Eintreten bestimmter Ereignisse aus dem Sleep Mode für die Verarbeitung von Daten zu erwecken. Bei den Events handelt es sich um Low-Level-Funktionen, welche das Wake-Up-Verhalten der Knoten beeinflussen und direkt in den Betriebssystemkern integriert werden mussten.

Anwender sind im Allgemeinen nicht an detaillierten Konfigurationen zur Optimierung eines Systems interessiert. Um dem Nutzer eine einfache Einflussnahme zu ermöglichen, bietet ACQL die Möglichkeit der Definition einer gewünschten minimalen Lebenszeit (`LIFETIME`-clause) des Sensornetzwerkes. Entsprechend der Vorgabe werden bei der Analyse der Anfrage Parameter für das Sampling und das Versenden von Datenpaketen so gewählt, dass die Lebenszeitvorgabe weitestgehend erfüllt wird. TinyDB führt eine einfache kostenbasierte Anfrageoptimierung auf der Basisstation aus. Anschließend wird die generierte Anfrage in ein einfaches Binärformat umgewandelt und über den Wurzelknoten im WSN propagiert. Bei der Anfrageoptimierung werden im Wesentlichen die Kosten für das Senden von Daten, den Sleep Mode und das Sampling berücksichtigt.

Sowohl Cougar als auch TinyDB verfügen über einen Metadaten-Katalog, welcher Statistiken über gemessene Werte enthält. Aufgrund der starken Verknüpfung von TinyDB mit dem Routing beinhaltet dessen Metadaten-Katalog zusätzlich noch Informationen über Nachbarknoten.

Im Umfeld von TinyDB erfolgte eine Vielzahl von weiteren Entwicklungen, wobei nahezu alle Komponenten von TinyDB auf eine energieffiziente Verarbeitung hin optimiert wurden (zum Beispiel Optimierung von einfachen Langzeitanfragen [191], die optimale Platzierung von Operatoren (Joins und Filter) innerhalb der Routinghierarchie [194] oder verteilte Joins [3] usw.).

2.2 Datenstromverarbeitung

Die Verarbeitung von kontinuierlichen, prinzipiell unendlichen Datenströmen ist mit einer Vielzahl von neuen Herausforderungen verbunden. [6, 21, 78] liefern eine umfangreiche Übersicht über die mit der Verarbeitung von Datenströmen verbundenen Probleme und bisher entstandene Lösungen. Im Folgenden soll ein kurzer Überblick über wesentliche Techniken gegeben werden.

2.2.1 Grundlagen

Da die komplette Speicherung von Datenströmen aus verschiedenen Gründen (beschränkter Speicherplatz bzw. Ergebnisse in Echtzeit) nicht möglich ist, wurden in den letzten Jahren bestehende Analyseverfahren an das Datenstromparadigma angepasst bzw. neu entwickelt. Die vorgestellten Lösungen lassen sich in Bezug auf ihren Umgang mit eingehenden Daten in zwei Klassen kategorisieren: Datenverdichtungen und fensterbasierte Verfahren. Nachfolgend soll ein Überblick über die entsprechenden Techniken gegeben werden.

Datenverdichtungen Das Ziel von Verfahren mit Datenverdichtung ist die Reduktion der Datenmenge unter Erhaltung deren charakteristischer Eigenschaften. Analysen werden anschließend stellvertretend auf den reduzierten Daten durchgeführt. Bei den Datenverdichtungen werden zwei grundlegende Herangehensweisen unterschieden: (i) Verfahren, welche repräsentative Werte aus dem Datenstrom für die Analysen einsetzen und (ii) Verfahren, welche die kompletten Daten in Form von Datenzusammenfassungen (*Synopsen*) aufbewahren.

Verfahren der ersten Gruppe reduzieren lediglich die Anzahl der Elemente des Datenstromes, wobei die Originalwerte erhalten bleiben. Die bekanntesten Mechanismen für das Finden geeigneter Teilmengen sind das *Sampling* (Auswahl von Vertretern) und das *Load Shedding* (Verwerfen von Teilsequenzen) [23, 201]. Die größte Herausforderung hierbei ist die Identifikation statistisch signifikanter Vertreter, welche als Repräsentanten dienen. Ein weiteres Problem ist die Behandlung von Ausreißern oder Bursts, welche, sofern sie nicht erkannt werden, die Menge der Repräsentanten verfälschen können.

Im Gegensatz dazu versucht man bei Datenzusammenfassungen lediglich die Eigenschaften der Daten des Stromes zu erfassen. Die Synopsen selbst enthalten dann zumeist keine Elemente des Originalstromes mehr. Die Zusammenfassungen sind dabei das Ergebnis der Anwendung von Funktionen auf den Datenstrom. Typische Funktionen sind einfache Aggregatfunktionen (zum Beispiel Summe, Durchschnitt, Minimum, Maximum, Variance) oder komplexere Funktionen wie zum Beispiel Histogramme [49, 93] oder Wavelets [86, 157, 209]. Aufgrund des zusammenfassenden Charakters von Synopsen, können Anfragen nur noch approximativ beantwortet werden. Rückschlüsse auf die Originaldaten sind nicht mehr möglich.

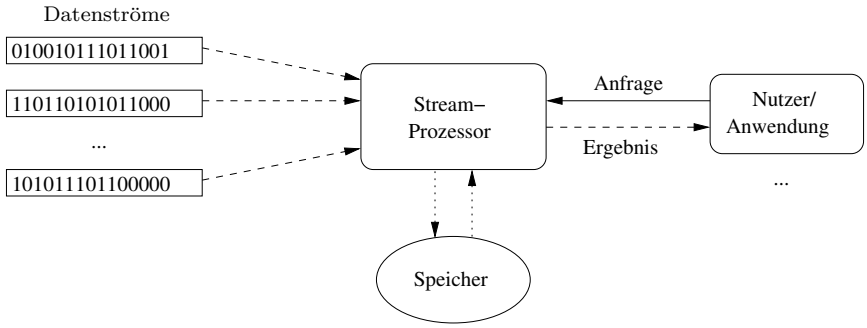


Abbildung 2.2: Generale Datenstromverarbeitung in DSMS [30]

Fenster Eine der größten Herausforderungen bei der Verarbeitung von Datenströmen ist der Umgang mit blockierenden Operatoren. Unter einem blockierenden Operator versteht man einen Operator, welcher den vollständigen Datenstrom zur Verarbeitung voraussetzt. Im Umfeld von DBMS sind dies zum Beispiel die verschiedenen Aggregationsoperatoren oder der Verbund. Eine einfache Lösung zur Vermeidung von blockierenden Operatoren ist der Einsatz von Fenstern (*windows*), bei welchen es sich im Wesentlichen um Datenspeicher mit einem zeitlichen Horizont handelt. Je nach Verfahren werden unterschiedliche Arten von Fenstern eingesetzt. Im Folgenden sollen die bekanntesten präsentiert werden.

Bei einem *Sliding Window* werden ausschließlich die letzten (aktuellsten) i Werte des Datenstromes gespeichert [61]. Beim Eintreffen eines neuen Wertes wird der älteste Wert aus dem Fenster entfernt. Im Gegensatz dazu werden beim *Landmark Window* alle Werte bis zu einer definierten Marke zwischengespeichert [21]. Das *Jumping-Window*-Modell [231] kann im weitesten Sinne als eine Kombination aus Landmark und Sliding-Window-Modell betrachtet werden. Hierbei gleitet das Fenster in Sprüngen über den Datenstrom, wobei die Sprungweite üblicherweise weiter ist als nur ein einzelnes Element. Ein weiteres, besonders speichereffizientes Fenster-Modell sind die *Tilted-Time Windows* (TTW) [51]. TTW's basieren auf der Idee, dass der Anwender häufig nur an den aktuellsten Ereignissen detailliert interessiert ist, ältere Daten dagegen in einem weniger detaillierten Grad benötigt werden. Als Beispiel führen die Autoren das *Natural Tilted-Time Window* an, bei welchem der Detailierungsgrad auf Basis der Zeit zusehends abnimmt. Beim *Logarithmic Tilted-Time Window* [84] erfolgt die Datenzusammenfassung einer logarithmischen Zeitskala entsprechend. Hierbei sind die letzten beiden Zeitpunkte exakt vorhanden. Weiter zurückliegende Werte werden je nach Alter in einem Zeitslot zusammengefasst. Die Menge der zu einem Wert zusammengefassten Daten folgt dabei dem Logarithmus, d.h. die in einem Slot zusammengefassten Werte werden kontinuierlich verdoppelt. So finden sich in den letzten Zeitslots Zusammenfassungen über $(1), 1, 2, 2, 4, 4, 4, 4, \ldots$ Werte.

2.2.2 Datenstrom-Managementsysteme

DSMS stellen eine konsequente Weiterentwicklung traditioneller DBMS dar. Sie erlauben dem Anwender die Definition von deskriptiven Anfragen über zuvor registrierten Datenströmen. In den letzten Jahren wurde eine Vielzahl von unterschiedlichen DSMS entwickelt (*NiagaraCQ* [50], *Nile* [99], *CAPE* [180], IBM's SPC [12, 81]), deren Funktionsweise sich intern teils stark voneinander unterscheidet. Allen gemein ist der prinzipielle Verarbeitungsablauf. Ein Anwender bzw. ein Anwendungsprogramm startet eine Anfrage, welche anschließend vom Streamprozessor verarbeitet und ausgeführt wird. Der Anwender erhält dann einen kontinuierlichen Strom von Ergebnissen (siehe Abbildung 2.2). Laufende Anfragen können dabei Teile der eingehenden Daten im Hauptspeicher zwischenhalten. Eine Auslagerung auf Sekundärspeicher ist zumeist aus Performancegründen nicht sinnvoll.

Im Folgenden sollen verschiedene DSMS, welche die Forschung in diesem Bereich signifikant beeinflusst haben, im Detail vorgestellt werden.

STREAM Eines der ersten Systeme, welches speziell an die Anforderungen von Datenströmen angepasst wurde, war STREAM (*Stanford Stream Data Manager*) [13, 14]. Die Autoren führen mit STREAM die beiden abstrakten Datentypen Strom und Relation ein. Zusammen mit den Abbildungen zwischen diesen bilden sie die Basis für die abstrakte Semantik von STREAM. Ströme werden in STREAM in Datenbank-Relationen (mittels Fenster-Operatoren) umgewandelt. Auf diesen Relationen können anschließend Standardoperationen für Relationen eingesetzt werden. Soll das Ergebnis wiederum ein Datenstrom sein, dann müssen die Relationen abschließend in einen Ergebnisstrom zurücktransformiert werden (siehe Abbildung 2.3(a)).

Im Rahmen des STREAM-Projektes wurde u.a. CQL (*Continous Query Language*) entwickelt, eine auf SQL basierende Anfragesprache, welche um die notwendigen Operatoren zur Manipulation von Relationen und Strömen erweitert wurde [13, 15, 16]. Hier sind insbesondere die Operatoren für *Stream-To-Relation*, *Relation-To-Relation* und *Relation-To-Stream* zu erwähnen, welche für die Transformierung des Datenstromes in Relationen und zurück zuständig sind.

Im Zusammenhang mit der Entwicklung von STREAM entstand auch eine Vielzahl von Techniken zur Optimierung der Anfragebearbeitung in einem DSMS. Hierzu gehören u.a. Load Shedding [23], Operator Scheduling [20] und die Reorganisation von Teilen von Anfragen [25]. Um grundlegende Funktionen von DSMS untereinander vergleichen zu können, wurde im Zuge von STREAM der CQL-basierte Benchmark Linearroad[2] entwickelt.

Aurora/Boralis Ein weiteres DSMS-Projekt ist Aurora [46]. Bei AURORA fließen die Elemente des Datenstromes durch einen zyklenfreien, gerichteten Graphen von Operatoren. Eines der besonderen Merkmale von AURORA ist, dass für jeden Operatorpfad

[2] http://www.cs.brandeis.edu/∼linearroad/

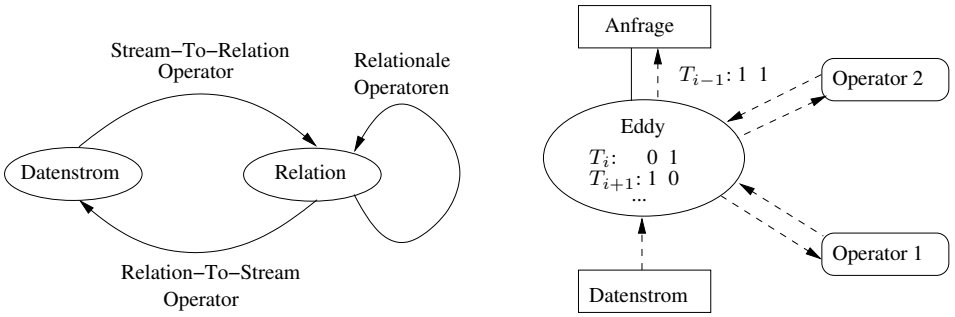
ein Wert (*utility function*) für die Qualität der Verarbeitung (QoS) mit angegeben wird. Das System platziert auf Basis dieser Werte Load-Shedding-Operatoren innerhalb der laufenden Anfragen, so dass das System den gewünschten Durchsatz bei bestmöglicher Qualität erzielt. Für ein dynamisches Load Shedding ermittelt AURORA vorab sogenannte *load shedding road maps* (LRSM), welche eine Menge von Alternativplänen enthalten, die mehr oder weniger (intensiv) Tupel verwerfen. Die LRSM's können in Abhängigkeit der aktuellen Auslastung zur Laufzeit vom System ausgetauscht werden.

Im Gegensatz zu STREAM, bei welchem eine deklarative Anfragebeschreibung erfolgt, werden bei Aurora Anfragepläne durch eine grafische Benutzeroberfläche definiert, welches dem *Box-and-arrow*-Konzept folgt. D.h. der Nutzer verbindet Operatoren (Boxen), durch welche die Tupel fließen. Hat der Nutzer seine Anfrage definiert, dann versucht der Optimierer durch ein Neuanordnen bzw. Kombinieren der Operatoren die beschriebene Anfrage zu optimieren.

Im Jahr 2005 ist das Aurora-Projekt zusammen mit dem Medusa-Projekt [52, 226] des MIT in dem Projekt Borealis [1, 11] aufgegangen. Der Schwerpunkt von Borealis liegt auf der verteilten Datenstromverarbeitung. Eine Instanz von Borealis kann dabei als ein riesiges Netzwerk aus Datenstromoperatoren angesehen werden. Jeder teilnehmende Knoten in einem Borealis-Netzwerk fungiert als Datenstrom-Server. Über entsprechende Proxyserver lassen sich auch WSNs in Borealis integrieren. Dem Proxyserver kommen dabei drei wesentliche Aufgaben zu. Er sammelt Statistiken über das angebundene WSN, so dass er vom DSMS-Teil vorgeschlagene Operatoren in Abhänggkeit der verfügbaren Ressourcen ablehnen bzw. Änderungen an den INQP-Teil durchreichen kann. Falls eine Verarbeitung im INQP stattfinden soll, entscheidet der Proxyserver, welche Implementierung eingesetzt wird, insofern alternative Implementierungen zur Verfügung stehen. Weiterhin soll der Proxyserver zusammen mit dem DSMS-Teil die Anfrage bezüglich QoS optimieren [2]. Prinzipiell stellt der Proxy somit die Schnittstelle zu beliebigen INQP her.

In [143] geben die Autoren Abschätzungen zur voraussichtlichen Laufzeit und zum Durchsatz des Systems, wobei sie sich auf die Aggregation und den statischen Verbund (Verbund von Datenstrom-Tupeln mit einer statischen Relation) beschränken. Aufgrund der Verwendung von TinyDB unterliegt dieser Ansatz den gleichen Problemen wie TinyDB, d.h die Menge der möglichen auszulagernden Operatoren ist stark beschränkt. Die teilweise bzw. vollständige Verarbeitung komplexer Operatoren im WSN wird prinzipiell nicht unterstützt.

TelegraphCQ *TelegraphCQ* [47, 48] ist ein an der Universität Berkeley entwickeltes System, welches auf dem Open Source DBMS PostgreSQL aufsetzt. Aufgrund der Verwendung eines bestehenden Datenbanksystems konnte bei *TelegraphCQ* eine Vielzahl von Komponenten wiederverwendet werden (zum Beispiel die Parser-Komponente und der Anfrageoptimierer). Zudem wurden kombinierte Anfragen über Datenströmen und statischen Datenbanken bereits aufgrund der Architektur leicht möglich. Eine Besonderheit von *TelegraphCQ* sind die *Eddies* [19]. Bei *Eddies* handelt es sich um Module zur Kontrolle des Tupelflusses. Ein Eddie entscheidet selbstständig, von welchem Operator



(a) STREAM [13] (b) Eddies in TelegraphCQ

Abbildung 2.3: Konzepte der Datenstromverarbeitung in DSMS

(innerhalb eines Blocks) ein Tupel als nächstes bearbeitet werden soll. Die Entscheidung erfolgt dabei auf Basis der Operatorenauslastung und dem von dem Tupel bereits besuchten Operatoren. Jedes Tupel führt zwei Bitmasken mit, welche einen Überblick über die bereits besuchten Operatoren und über die noch abzuarbeitenden liefern. Abbildung 2.3(b) zeigt ein Beispiel für die Tupel-Verarbeitung mittels Eddies.

In der Literatur wurden verschiedene mögliche Routing-Strategien für Eddies vorgestellt. Manche von diesen Strategien betrachten systemspezifische Eigenschaften wie zum Beispiel die Ausführungskosten eines Operators pro Tupel oder den dafür notwendigen Speicherplatz (*Ticket Scheduling* [19], *batch-basiertes Scheduling* [64]). Andere wie zum Beispiel das *content-basierte Routing* [35] beziehen Tupeleigenschaften (den Zusammenhang zwischen Attributen und deren Selektivitäten in den Operatoren) in den Entscheidungsprozess mit ein.

WaveScope Wavescope [87] ist ein junges Projekt, welches sich der effizienten Verarbeitung von Signalen aus Sensordatenströmen verschrieben hat. Die wesentlichen Teile der Datenverarbeitung finden dabei auf einer zentralen Instanz statt. Der größte Unterschied des Systems besteht in dem Verzicht auf gleitende Fenster. Statt dieser setzt WaveScope auf so genannte Signal-Segmente (SigSeg). Ein SigSeg entspricht dabei einem Fenster fester (Bit-)Länge, welche von den Operatoren verarbeitet wird (ähnlich den Jumping Windows). Die Besonderheit dabei ist der vollständige Verzicht auf Zeitstempel innerhalb eines SigSeg. D.h. alle Werte eines gemeinsamen SigSeg werden dem selben Zeitpunkt zugeordnet, so dass der üblicherweise zusätzliche Zeitstempel entfällt. Der damit verbundene Ordnungsverlust der Daten innerhalb eines SigSeg wird zugunsten der Performance in Kauf genommen. WaveScope unterstützt weiterhin die Einbindung von Sensornetzwerken, wobei sich aber im Wesentlichen auf das Anhäufen von Daten in SigSegs beschränkt wird. Diese werden anschließend an die zentrale Komponente gesendet und von dieser weiterverarbeitet.

Eine weitere Besonderheit von WaveScope ist WaveScript [87, 88]. Bei WaveScipt handelt es sich um eine Spracherweiterung ähnlich den aus den Datenbanksystemen be-

kannten UDF's (*User Defined Functions*). Mit dieser lassen sich auf einfache Weise neue Operatoren auf Grundlage einfacher Algorithmen kreieren.

Weitere DSMS Das an der Universität Marburg entwickelte *PIPES* [133, 134] unterscheidet sich von anderen Systemen insoweit, dass es selbst kein eigenständiges DSMS implementiert. PIPES stellt eine Infrastuktur (Bibliothek von Operatoren) zur Verfügung, auf deren Basis sich beliebige, speziell an die Anwendungsbedürfnisse angepasste DSMS entwickeln lassen. Neben der Operatorbibliothek bietet es zudem verschiedene alternative Laufzeitkomponenten (Scheduler, Speichermanager, Anfrage-Optimierer). PIPES wurde vollständig in Java implementiert und ist Teil der XXL-Library [42,59]. Ein ähnliches Projekt ist Odysseus [36], welches sich sich im Wesentlichen durch die Vielzahl unterstützter Datenmodelle von PIPES unterscheidet.

Das an der TU Dresden entwickelte QStream [185, 186] wurde mit dem Ziel einer Echtzeit-Bearbeitung von Anfragen entwickelt. Zu diesem Zweck baut QStream auf einem Echtzeit-Betriebssystem auf, welches die einzelnen Operatoren einer Anfrage periodisch aktiviert. Der Anwender ist in der Lage, für jede Anfrage eine minimal notwendige Ergebnisgüte (QoS) zu definieren. Um die Echtzeitanforderung zu gewährleisten kann das System einen zusätzlicher Sampling-Operator einfügen, welcher bei Bedarf Daten des Stromes verwirft, solang das geforderte QoS-Level nicht unterschritten wird.

Gigascope [57] ist ein DSMS, welches zur Überwachung der Netzwerkauslastung konzipiert wurde. Bei Gigascope wird insbesondere auf das Problem der extrem hohen Datenankunftsraten eingegangen. Gigascope unterscheidet dabei zwei Anfrageebenen: (i) Low-Level-Anfragen überwachen direkt das Netzwerkinterface und (ii) Higher-Level-Anfragen dienen der Weiterverarbeitung der Ergebnisse von (i). Die Entscheidung, in welchen Bereich eine Anfrage fällt, wird bei der Anfrageanalyse getroffen.

Ein interessantes System, welches im Zusammenhang mit DSMS erwähnt werden soll, ist RapidMiner [160] (früher YALE) der Firma Rapid-I. Hierbei handelt es sich im eigentlichen Sinne nicht um ein DSMS. RapidMiner ist eine Open-Source-Anwendung, welche dem Nutzer einen schnellen Zugang zu Techniken u.a. des Data Mining erlaubt. Im Wesentlichen legt der Anwender hierbei Datenfluss-Graphen an, bei denen die Knoten die einzelnen Operatoren repräsentieren. Bei den Operatoren handelt es sich oftmals um Implementierungen der bekanntesten Verfahren aus den Bereichen Data Mining, Knowledge Discovery und Machine Learning. Beim Verarbeitungsprozess „fließen“ die Ergebnisse eines Operators in den nächsten, wobei zumeist alle Tupel einen Operator passiert haben müssen, um an den nächsten weitergeleitet werden zu können. Über Plugins lässt sich die Funktionalität des RapidMiners leicht erweitern (zum Beispiel lassen sich so die Weka-Tools der Universität Waikato, eine Bibliothek an Werkzeugen für die Data-Mining-Analyse, einfach nachinstallieren). Echte Data-Stream-Mining-Funktionalität kann über entsprechende Plugins ebenfalls nachgerüstet werden.

Ubiquitous Data Stream Mining Ein motiviert durch die Miniaturisierung der Hardware immer interessanter werdendes Thema ist das *Ubiquitous Data Stream Mining* [4, 77]. Unter diesem Begriff fasst man informationsverarbeitende Systeme zusammen, welche sich direkt und immer im Umfeld des Nutzers befinden, zum Beispiel PDA's, Mobiltelefone oder in Kleidung integrierte Hardware. Die Herausforderungen im Bereich des Ubiquitous Computing ähneln dabei denen der drahtlosen Sensornetzwerke (beschränkter Energiespeicher und Miniaturisierung). Der Schwerpunkt ist hierbei aber weniger auf der Ebene der Kommuikation der Hardware untereinander zu sehen, als vielmehr in der Kommunikation mit dem Nutzer. Für entsprechende Systeme wie zum Beispiel *MobiMine* [122] und VEDAS [121, 145] bedeutet dies, dass Anwendungen nach dem Client/Server-Paradigma entwickelt werden und der Client im Wesentlichen der Visualisierung von Ergebnissen bzw. der Vorverarbeitung von einfachen Aufgaben dient.

2.2.3 Integrierte DSMS/INQP-Systeme

Mit HiFi [55] präsentierten Cooper et al. ein System, welches die Vorteile aus einem DSMS und einem INQP miteinander verbinden soll. Hierzu kombinieren sie das DSMS TelegraphCQ und den INQP TinyDB. TinyDB wird bei HiFi zum Einsammeln und Verdichten (Aggregieren) der gemessenen Sensordaten eingesetzt. Komplexe Operationen (Korrelation, Ausreißerdetektion) werden von TelegraphCQ übernommen.

Ein HiFi-Netzwerk entspricht einem Mehrwege-Baum, dessen Blätter drahtlose Sensornetzwerke sind. D.h. ein HiFi-Netzwerk kann beliebig viele WSNs in den Verarbeitungsprozess integrieren. Jeder der inneren Knoten dieses Baumes entspricht einer Instanz von TelegraphCQ. Der Wurzelknoten liefert letztendlich den Ergebnisstrom. Dieser hierarchische Aufbau erlaubt die gleichzeitige Verarbeitung hoher Datenmengen von räumlich verteilten Datenquellen. Die inneren Knoten müssen dabei den hohen Leistungsanforderungen durch entsprechende Hardware genügen. Im Allgemeinen ist der Wurzelknoten der am stärksten beanspruchte Knoten. Je nach Datenaufkommen und auszuführenden Operationen kann sich dieser Schwerpunkt aber auch auf einen der inneren Knoten verlagern. Entsprechende Anpassungen hierauf sind in HiFi nicht vorgesehen.

2.2.4 Föderierte Datenbanksysteme

Das Zusammenführen von Daten aus autonomen (heterogenen) Datenquellen ist nicht neu. So bieten föderierte Datenbanksysteme spezielle Konzepte zum Verbinden von Daten aus verschiedenen unabhängigen Quellen, ohne diese zuvor in ein einheitliches Datenbanksystem zu integrieren. Der Schwerpunkt von entsprechenden Systemen wie zum Beispiel IBM's Garlic [44,206] liegt in der Schaffung einer einheitlichen Schnittstelle zu den verteilten Datenbeständen. Die Schemata der einzelnen Systeme werden dabei nicht angepasst, so dass existierende Anwendungen ohne Änderungen weiter verwendet werden können [195].

Ein interessantes Projekt, welches dieses Konzept auf DSMS ausweitet ist, MaxStream [39]. MaxStream integriert mittels Wrapper Datenströme aus unterschiedlichen DSMS (zum Beispiel Coral8 und StreamBase[3]) und bietet dem Anwender eine einheitliche Schnittstelle für die Definition von Anfragen und den Zugriff auf die Ergebnisse. Kern von MaxStream ist dabei der Federator, welcher als Schnittstelle zwischen den DSMS und den Clientanwendungen dient. Der Federator wurde als eine Erweiterung des relationalen DBMS SAP MaxDB[4] umgesetzt, so dass dessen SQL-Fähigkeiten sowie das Transaktionsmanagement übernommen werden konnten.

Prinzipiell lässt sich die Integration verschiedener WSN in die Verarbeitung eines DSMS auch als ein föderiertes System betrachten. Aufgrund der Autonomie der WSN wäre mit diesem Ansatz aber keine plattformübergreifende Anfrageoptimierung möglich, weswegen dieser Ansatz im Weiteren nicht verfolgt werden soll.

2.3 Data Stream Mining

In der Literatur findet sich eine Vielzahl von Definitionen für den Begriff „Data Mining“. Nach [72] wird das Data Mining als Teil der „Knowledge Discovery in Databases“ (KDD-Prozess) angesehen, wobei unter KDD der „*nichttriviale* Prozess der Identifizierung gültiger neuer, potentiell nützlicher und schlussendlich verständlicher Muster in Daten“ verstanden wird [173]. Unter Data Mining als Teilprozess des KDD versteht man dabei die Anwendung von Analyseverfahren zur Identifikation von Mustern oder Modellen in Daten.

Die zwei wesentlichen Ziele des Data-Mining-Prozesses sind die Vorhersage von unbekannten oder zukünftigen Werten und die Beschreibung von Mustern, so dass diese für den Nutzer interpretierbar werden. Die Grenzen zwischen beiden Prozessen können dabei u.U. ineinander übergehen. Das Ziel der Vorhersage und Beschreibung kann durch vielfältige Methoden erreicht werden. Fayyad et al. [72] führen die folgenden Methoden an: Klassifikation, Regression, Clustering, Erstellen von Zusammenfassungen, Finden von Abhängigkeitsmodellen und Änderungserkennung.

Im Folgenden sollen verschiedene Verfahrensklassen im Detail beschrieben werden, wobei insbesondere auf Entwicklungen im Datenstromkontext eingegangen werden soll.

Clustering Unter Clustering versteht man das (nicht überwachte) Gruppieren von Objekten in verschiedene, vorher nicht bekannte Klassen, so dass Objekte einer Klasse sich möglichst ähnlich sind und Objekte verschiedener Klassen sich möglichst stark voneinander unterscheiden. Zusätzlich zu diesem Ziel betrachtet man oftmals auch das Problem der hierarchischen Einordnung von Clustern (Cluster enthalten Teilcluster). Für das Clustern auf statischen Daten existiert eine Vielzahl verschiedener Lösungen (zum Beispiel partitionierende [103], hierarchische oder dichtebasierte Verfahren [115]).

[3] http://www.streambase.com/
[4] http://www.sdn.sap.com/irj/sdn/maxdb

Da die klassischen Verfahren für die Analyse zumeist das Vorhandensein der vollständigen Daten voraussetzen, sind sie für die Verarbeitung von Datenstromdaten ungeeignet. In den letzten Jahren wurden daher verschiedene One-Pass-Verfahren für Datenströme entwickelt [94]. Zu den bekanntesten Verfahren zählen STREAM [94], BIRCH [227] und CLUStream [7,8]. Die existierenden Verfahren verfolgen dabei zwei wesentliche Ansätze: Die erste Klasse von Verfahren ordnet beim Eintreffen eines neuen Datenpunktes diesen in einen existierenden Cluster ein. Die zweite Klasse von Verfahren teilt den Datenstrom in Blöcke auf. Auf diesen wird dann das Clusterverfahren angewandt. Anschließend werden die so gewonnenen Teilcluster mittels des Verfahrens rekursiv zu den finalen Clustern vereinigt.

In der Literatur finden sich zahlreiche In-Network-Cluster-Verfahren. Das Clustering dient hierbei zumeist aber nicht der Wissensfindung mit anschließender Übermittlung an die Basisstation, sondern dem Aufbau effizienter Routing-Topologien [225]. So werden beispielsweise beim Spatial Clustering in [158] Cluster von Sensoren im Netzwerk gesucht, welche ähnliche Daten erfassen. Anschließend ist es ausreichend, wenn ein Knoten eines Clusters Daten erfasst und versendet. Die anderen Knoten eines Clusters können das Messen und Versenden ihrer Daten daraufhin weitestgehend einstellen, so dass an dieser Stelle entsprechend Energie gespart wird.

Klassifikation Die Klassifikation zählt zu den überwachten Lernverfahren, deren Ziel es ist, aus erlerntem Wissen (dem Modell) eine Klassenzuordnung für neue Datenpunkte vornehmen zu können. Die Klassifikation findet in zwei Phasen statt. Während der Trainings- oder Lernphase wird das Modell gelernt. Im statischen Szenario erfolgt dies üblicherweise anhand von Trainingsdaten. Während der zweiten Phase werden die neuen Daten klassifiziert. Das Finden eines möglichst guten Modells stellt die größte Herausforderung auf diesem Gebiet dar. Für die Klassifikation auf statischen Datensätzen gibt es eine Vielzahl von Verfahren, zum Beispiel instanzbasierte Verfahren, Entscheidungsbäume, Regellerner oder neuronale Netze [198].

Aufgrund der notwendigen Trainingsphase scheint das Problem der Klassifikation mit dem Datenstrom-Paradigma unvereinbar. Um das Modell kontinuierlich anpassen zu können, müssen alle Stromdaten als Trainingsdaten betrachtet werden, was eine anschließende Klassifikation wiederum überflüssig machen würde. In Abhängigkeit vom Klassifizierer lassen sich dennoch Szenarien entwickeln, in denen eine Klassifikation auf Datenströmen sinnvoll ist [30, 119].

Frequent Pattern Mining Das Finden häufiger Muster ist eine Grundvoraussetzung zur Lösung vieler Data-Mining-Probleme wie zum Beispiel des Assoziation Rule Mining, des Sequential Pattern Mining oder des Closed Pattern Mining. Erstmals für Datenbanken beschrieben wurde das Problem von Agrawal et al. in [9]. Zu den bekanntesten Lösungsverfahren zählt der Apriori-Algorithmus [9, 10]. Die weitere Forschung im Bereich des Frequent Pattern Mining beschäftigte sich u.a. mit der Entwicklung performanter Verfahren sowie der Entwicklung datenstromgeeigneter Lösungen (Lossy

Counting [152]). Eine Möglichkeit der kompakten Repräsentation von häufigen Mustern sind Präfix-Bäume. Verfahren wie zum Beispiel DSM-FI [142]) oder FP-Stream [84] verwenden entsprechende Synopsen zur effizienten Speicherung.

Ein möglicher Ansatz zum In-Network Frequent Pattern Mining wird in [179] präsentiert. Hierbei kann der Nutzer zunächst definieren, welche Ereignisse auf häufige Muster hin überwacht werden sollen. Die Anfrage wird anschließend von der Basisstation in ausführbaren Code übersetzt und auf die Sensorknoten verteilt. Die Knoten überwachen daraufhin die Ereignisse einer beschränkten Nachbarschaft und überprüfen diese auf häufig auftretende Muster. Beim Pattern-Mining-Verfahren handelt es sich um eine Datenstromversion des Apriori-Algorithmus (der Datenstrom wird in Batches zerlegt und diese werden anschließend getrennt voneinander durch das Apriori-Verfahren ausgewertet).

Prinzipiell können beim Data Stream Mining zwei grundsätzlich verschiedene Ansätze zum Umgang mit den zu analysierenden Daten betrachtet werden:

- Alle Elemente eines Stromes werden einzeln als Objekte betrachtet, d.h. die Analyse wird entlang der Datenstromelemente durchgeführt.
- Der Datenstrom selbst wird als sich veränderndes Objekt betrachtet.

Beim Clustering bedeutet dies zum Beispiel, dass im ersten Fall Cluster entlang der zeitlich getrennten Datenobjekte gesucht werden und im zweiten Fall über verschiedene Datenströme hinweg geclustert wird.

Aufgrund des zumeist modularen Aufbaus von DSMS (und der Kapselung von Algorithmen in Operatoren) lassen sich komplexe Data-Mining-Operatoren auf einfache Weise in existierende DSMS integrieren [31]. Beim Data Mining in WSN gestaltet sich dies schwieriger. Wurden entwickelte In-Network-Lösungen nicht nur auf Basis von Simulationen evaluiert, so ist deren Implementierung in der Regel auf ein spezielles WSN zugeschnitten und fester Teil der entwickelten Anwendung.

2.4 Fazit

Sowohl im Bereich der Datenstromverarbeitung als auch im Bereich der In-Network--Verarbeitung existieren bereits zahlreiche Lösungen. Aufgrund der Vielzahl von Verfahren in diesen Bereichen kann obiges Kapitel lediglich einen Überblick über die wichtigsten Entwicklungen in den letzten Jahren geben. Dennoch lassen sich anhand der vorgestellten Verfahren bereits einige offene Probleme ableiten, welche im Rahmen der vorliegenden Arbeit gelöst werden sollen.

- *Manuelles Netzwerk Deployment*: Alle oben beschriebenen INQP setzen für das Routing auf die Entwicklung vollständig autonomer Topologien. Die dafür notwendigen Algorithmen sind oftmals Bestandteil des WSN-Betriebssystems. Zusätzlich

zu diesen Strategien wurden zahlreiche Routing-Algorithmen entwickelt, welche die Aggregation der Daten in den Mittelpunkt stellen. Erste Ansätze, welche orthogonal zu diesen Verfahren arbeiten und einen manuellen Eingriff in den Aufbau geeigneter Topologien erlauben, wurden ebenfalls präsentiert. Allerdings unterstützt keines der in der Literatur gefundenen Verfahren eine vollständig manuelle Entwicklung logischer Netzwerktopologien, welche zum Beispiel für die Integration externen Wissens in den Verarbeitungsprozess notwendig sind.

- *Deklarative Anfragebeschreibung*: Viele der vorgestellten INQP- bzw. DSMS-Lösungen erlauben das Stellen deklarativer Anfragen. Hierbei beschreibt der Anwender lediglich die zu erreichenden Ziele. Wie diese erreicht werden, wird automatisch vom verwendeten System bestimmt. Die Entscheidung der optimalen Ausführungsumsetzung sollte dabei immer im Interesse aller Systemkomponenten gefällt werden. Aufgrund des geringen Funktionsumfanges aktueller INQP bzw. der nur rudimentären Integration der INQP-Optimierung in existierende kombinierte DSMS/INQP-Lösungen ist eine vollständig autonome plattformübergreifende Optimierung häufig nicht gewährleistet.

- *Ausdrucksmächtigkeit*: Die oftmals geringen Speicherressourcen auf den Sensorknoten stellen für den Funktionsumfang existierender INQP-Systeme häufig eine der wesentlichen Beschränkungen dar. Neben der für die Anfrageverarbeitung notwendigen Verarbeitungsebene werden in der Regel lediglich grundlegende Funktionen wie zum Beispiel Aggregation, Filtern und Verbund unterstützt. Die Einschränkungen gehen so weit, dass zum Beispiel in TinyDB lediglich ein Verbundoperator (Nested Loop Join) integriert wurde, obwohl weitaus effizientere Lösungen für spezielle Anwendungsfälle bekannt sind. Alle diese alternativen Implementierungen gleichzeitig auf den Sensorknoten bereit zu stellen, ist aufgrund der geringen Ressourcen der Sensorknoten zumeist nicht möglich, aber wünschenswert.

- *Komplexe Algorithmen*: Neben den primitiven In-Network-Verfahren existiert auch eine Vielzahl von komplexen Algorithmen (zum Beispiel aus dem Bereich Data Mining), für welche zusehends auch Anpassungen an verteilte Systemen entwickelt werden. Die Verarbeitung findet dabei teilweise oder vollständig im WSN statt. Zumeist handelt es sich bei den entwickelten Ansätzen um Insellösungen, welche lediglich in prototypischer Form zur Verfügung stehen und nicht Teil eines integrierten Systems sind. Welche Teile der Algorithmen von welcher Systemkomponente verarbeitet werden, muss hierbei oftmals explizit durch den Anwender bestimmt werden.

In den folgenden Abschnitten dieser Arbeit soll versucht werden die hier aufgedeckten Probleme als Teil einer neuen Stream-Processing Engine zu Lösen.

Kapitel 3

Systementwurf

Im letzten Abschnitt wurden verschiedene Lösungen in den Bereichen der Datenstrom- und der In-Network-Verarbeitung von Anfragen beschrieben. Abschließend wurden verschiedene offene Probleme aufgezeigt, welche durch die bestehenden Lösungen nicht bzw. nur bedingt gelöst werden.

Das Ziel dieses Kapitels ist der Entwurf eines Systems, welches die im letzten Abschnitt beschriebenen Probleme löst. Zunächst werden hierzu detaillierte Anforderungen für das System spezifiziert. Basierend auf diesen Anforderungen wird anschließend ein konkreter Systementwurf abgeleitet, welcher der weiteren Arbeit zugrunde liegt. Zum Schluss soll in diesem Kapitel noch auf die Nutzer-System-Schnittstelle im Detail eingegangen werden.

3.1 Anforderungsanalyse

Der Schwerpunkt der vorliegenden Arbeit liegt im Entwurf und der Entwicklung eines DSMS mit partieller In-Network-Verarbeitung. Im nachstehenden Abschnitt werden funktionale Anforderungen an das System spezifiziert. Anschließend werden Bedingungen definiert, welche das Einsatzgebiet des Systems eingrenzen.

3.1.1 Anforderungen an das System

Ausdrucksmächtigkeit Eine der wesentlichen Einschränkung aktueller WSN-Lösungen ist deren zumeist beschränkter Funktionsumfang. Bei Sensorknoten handelt es sich um Kleinstcomputer, deren Möglichkeiten zum gegenwärtigen Zeitpunkt oftmals nicht vollständig ausgeschöpft werden. Prinzipiell können beliebig komplexe Algorithmen von den auf den Sensorknoten zu findenden Prozessoren ausgeführt werden. Insofern diese In-Network-Verarbeitung energetisch effizienter ist als das alternative Versenden der für eine externe Berechnung notwendigen Daten, sollte diese auch vom DSMS/INQP-System unterstützt werden.

Dem entgegen steht der zumeist stark begrenzte Speicherausbau der Sensoren, welcher nur ein beschränktes Repertoire an Algorithmen erlaubt. So existierte zum Beispiel ursprünglich der Verbund in TinyDB lediglich in Form eines einfachen Nested Loop Joins [146]. Aus den Bereichen der DBMS und der DSMS sind allerdings weit energieeffizientere Verfahren bekannt. Deren Einsatz ist aber oftmals nur bei bestimmten Anfragetypen (zum Beispiel Hashjoins) oder bei speziellen Datencharakteristika sinnvoll. Wird ein solcher Operator bei laufenden Anfragen nicht eingesetzt, dann belegt er ausschließlich Speicherplatz, welcher weiteren alternativen Operatoren somit nicht mehr zur Verfügung steht.

Das im Rahmen dieser Arbeit entwickelte System soll daher neben den Basisfunktionen, welche gegenwärtig von existierenden Lösungen bereits unterstützt werden, auch die Möglichkeit zur einfachen Integration weiterer Verfahren ermöglichen. Neben diesen zusätzlichen Verfahren soll das System die Option zur Auswahl alternativer Implementierungen für Operatoren bieten, welche bei entsprechender Verwendung zu einer Reduktion der für die Verarbeitung von Anfragen notwendigen Energie beitragen. Sind bestimmte Implementierungen für die Ausführung laufender Anfragen nicht notwendig, so sollen diese auch keinen Speicher auf den Sensorknoten belegen. Dieser steht dann weiteren Anfragen oder Operatoren zur Verfügung.

Komplexe Operatoren Wie bereits festgestellt, existieren bisher keine INQP, welche die Verarbeitung von komplexen Data-Mining-Verfahren unterstützen. Zwar existieren WSN-Lösungen für die verschiedensten Data-Mining-Probleme, allerdings findet deren Entwicklung zumeist unabhängig von der Verwendung in INQP statt. Somit ist für deren Einsatz immer entsprechendes Expertenwissen notwendig bzw. für eine pre- bzw. post-Verarbeitung der Sensordaten müssen die notwendigen Verfahren immer wieder neu entwickelt werden.

Ein weiteres Problem ist die oftmals hohe Komplexität der verwendeten Data-Mining-Verfahren. So kann es zum Beispiel notwendig sein, in Abhängigkeit der Dateneigenschaften und des verwendeten Algorithmus bestimmte Teile des Verfahrens auf eine zentrale, von der Energiebilanz des WSN unabhängige Komponente, auszulagern. Mit der Verarbeitung auf der zentralen Komponente fallen wiederum zusätzliche Energiekosten für den Datenversand an. D.h., Teile des komplexen Verfahrens werden zur Datenreduktion im WSN ausgeführt, die restliche Verarbeitung findet auf der zentralen Instanz statt.

Eines der wesentlichen Ziele des zu entwickelnden Systems soll daher die Unterstützung komplexer Verfahren sein. Hierbei steht wiederum das Ziel der energetischen Optimierung im Vordergrund. D.h., es sollen sowohl Algorithmen unterstützt werden, deren Verarbeitung vollständig im WSN stattfindet als auch Verfahren, welche teilweise auf einer zentralen Komponente bearbeitet werden. Welches der Verfahren in einer konkreten Situation zur Anwendung kommt, soll dabei dynamisch in Abhängigkeit vom Energieprofil der möglichen alternativen Ausführungspläne gefällt werden.

Deklarative Anfrageentwicklung Schwerpunkt des zu entwickelnden Systems ist die automatische Anfrageoptimierung. Der Nutzer soll in die Lage versetzt werden, lediglich das Anfrageziel zu beschreiben. Das System hat selbstständig zu entscheiden, welche Anfrageteile (in Abhängigkeit von definierten Nutzerpräferenzen) in welcher Form vom System ausgeführt werden.

Um einen möglichst optimalen Ausführungsplan zu entwickeln, muss die Anfrageverarbeitung und -optimierung dabei (einheitlich) über alle Systemkomponenten hinweg erfolgen. Im Zusammenhang mit dieser paradigmenübergreifenden Anfrageoptimierung ergeben sich weitere Probleme, die durch das zu entwickelnde System zu lösen sind:

- Es müssen Kostenmodelle entwickelt werden, welche die Kosten für den Einsatz alternativer Operatoren berücksichtigen. Weiterhin muss die Integration und die Kostenentwicklung bei komplexen plattformübergreifenden Operatoren betrachtet werden. Hier muss insbesondere auf das Problem der partiellen In-Network-Ausführung von Algorithmen eingegangen werden. D.h., welche Verarbeitungsteile eines komplexen Verfahrens werden auf welcher Komponente ausgeführt und welche Kosten fallen für deren Verarbeitung an?

- Die Menge möglicher physischer Pläne steigt mit der Zahl der Alternativen. Entsprechend aufwändig kann sich die Anfrageoptimierung gestalten. Das System muss über entsprechende Techniken verfügen, welche es erlauben, einen möglichst guten (nicht notwendigerweise den besten) Plan in akzeptabler Zeit zu ermitteln.

- Neben der Energieminimierung spielen bei der Anfrageoptimierung auch weitere Kriterien eine Rolle. Diese dürfen bei der Planauswahl nicht vernachlässigt werden. So wäre es zum Beispiel aus energetischer Sicht das effizienteste, das WSN komplett zu deaktivieren. Gleichzeitig würde aber die Ergebnisgüte auf ein nicht akzeptables Maß absinken. D.h., eine ausschließliche Betrachtung eines einzelnen Kriteriums, beispielsweise des Energieverbrauchs, ist ein nicht ausreichendes Optimierungsziel. Der zu entwickelnde Anfrageoptimierer des geforderten Systems muss daher neben der vom WSN benötigten Energiemenge weitere Optimierungsziele betrachten können und diese Faktoren hinreichend gut optimieren, so dass für den Anwender der bestmögliche Anfrageplan entwickelt wird.

Manuelle Entwicklung und Integration von Netzwerktopologien Mit der Möglichkeit, komplexe Verfahren im WSN zu bearbeiten, ergibt sich auch die Frage nach dem Ausführungsort der anzuwendenden Operatoren. Wie bereits erwähnt, ermöglicht das Zuführen von externem Wissen (zum Beispiel der Ort der Sensorknoten bzw. deren mögliches Verarbeitungspotential) teilweise erst eine energie-effiziente In-Network-Ausführung komplexer Operatoren. Bisher gibt es kein System, welches eine entsprechende manuelle Wissensanreicherung unterstützt.

Das zu entwickelnde System sollte daher neben Anfragen mit komplexen Operatoren auch die nutzerdefinierte Bestimmung der Ausführungslokalisation unterstützen. Zu die-

sem Zweck sind entsprechende Techniken zu integrieren, welche es dem Anwender erlauben, auf einfache und verständliche Weise entsprechende logische Netzwerkstrukturen zu entwickeln und diese im Rahmen der In-Network-Verarbeitung sinnvoll einzusetzen.

Verknüpfung von Informationen Oftmals sind gemessene Werte allein für eine sinnvolle Auswertung nicht ausreichend. Erst die Kombination mit zusätzlichen Informationen ermöglicht eine angemessene Analyse. Zu diesem Zweck ist das zu entwickelnde System so aufzubauen, dass Informationen aus weiteren Quellen mit den von den Sensoren des WSN ermittelten Daten auf einfache Weise verknüpft werden können. Die zugeführten Informationen können dabei sowohl statisch als auch dynamisch sein, also aus Datenströmen stammen.

3.1.2 Randbedingungen

Im Folgenden sollen einschränkende Bedingungen für den Einsatz des zu entwickelnden Systems gegeben werden.

Beschränkung der Basisstationen Üblicherweise sind WSNs so aufgebaut, dass von jedem teilnehmenden Knoten Nachrichten in das Netzwerk gesendet und Daten empfangen werden können. Diese Struktur erlaubt generell eine beliebige Anzahl an zentralen Instanzen (Basisstationen), welche den Zugriff auf das Netzwerk ermöglichen. Das zu entwickelnde System soll dahingehend eingeschränkt werden, als dass es nur eine einzige zentrale Instanz gibt. Lediglich diese eine zentrale Instanz verfügt über die notwendigen Informationen für die Anfrageoptimierung bzw. die Anpassung laufender Anfragen.

Für eine Aufhebung dieser Einschränkung (Zugriff auf das WSN über mehrere Basisstationen) müsste ein System zum Austausch der Status-Informationen unter den Basisstationen aufgebaut werden (ähnlich Borealis). Da es den Rahmen der vorliegenden Arbeit übersteigt, soll dies im Weiteren nicht betrachtet werden.

Unterstützte Anfragetypen Im Umfeld von Datenströmen können die folgenden Klassen von Anfragen auftreten bzw. werden in den unterschiedlichen Systemen unterstützt:

- *Kontinuierliche Anfragen*: Hierbei handelt es sich um Anfragen, die (während eines definierten Zeitraumes) fortwährend vom System ausgeführt werden. Diese Anfragen entsprechen den kontinuierlichen Anfragen in einem DSMS und liefern entsprechend einen Datenstrom zurück.
- *Event-Anfragen*: Diese Form der Anfragen überwacht spezielle Ereignisse. Treffen diese ein, dann werden weitere Verarbeitungsschritte eingeleitet. Ein Beispiel hierfür ist: „Wenn eine Bewegung im Raum detektiert wurde, dann mache ein Foto!“

- *Snapshot-Anfragen oder Ad-Hoc-Anfragen*: Dies sind Anfragen, welche sich nur auf den aktuellen Zeitpunkt beziehen. Ein Beispiel hierfür ist: „Welchen Temperaturwert misst Sensor X in diesem Moment?“

Das Ziel dieser Arbeit ist die Entwicklung eines Systems, welches kontinuierliche Anfragen auf Datenströmen ausführen kann. Bei event-basierten Anfragen handelt es sich im Wesentlichen um einen Spezialfall kontinuierlicher Anfragen. Anstelle eines einfachen Ergebisstromes werden jedoch gezielt Ereignisse ausgelöst. Prinzipiell wird dieser Anfragetyp also durch die Möglichkeit zur Verarbeitung kontinuierlicher Anfragen abgedeckt. Auf die Definition spezieller Events soll im Kontext dieser Arbeit nicht gesondert eingegangen werden.

Da bis zum Zeitpunkt der Anfragedefinition einer Ad-Hoc-Anfrage nicht bekannt ist, welche Operatoren benötigt werden, müssten somit alle Operatoren auf allen Sensorknoten zum Anfragezeitpunkt Teil der Laufzeitumgebung sein (wie in TinyDB) bzw. zum Anfragezeitpunkt auf die Sensorknoten transferiert werden. Eine Vorab-Installation aller möglichen Operatoren bzw. Alternativen auf den Sensorknoten ist nicht möglich bzw. schränkt den Optimierungsbereich des Gesamtsystemes zu stark ein (siehe Argumentation oben). Der alternative Ansatz, Kurzeit-Anfragen und die damit verbundenen Operatoren auf die Sensorknoten zu transferieren, ist mit so hohen Energie-Kosten verbunden, dass eine Unterstützung nicht sinnvoll erscheint und im Weiteren auch nicht betrachtet werden soll.

3.2 Systemarchitektur

Im weiteren Verlauf dieses Kapitels soll das kombinierte DSMS/INQP *AnduIN* [53, 125, 126] vorgestellt werden, welches, wie später gezeigt wird, den oben beschriebenen Anforderungen genügt. Die grobe Architektur des Systems ist dabei durch die in Abschnitt 3.1.2 beschriebenen Rahmenbedingungen definiert. So besteht *AnduIN* aus einer im WSN laufenden Komponente und einer zentral (zum Beispiel auf einem PC oder Server) auszuführenden Komponente. Abbildung 3.1 zeigt die prinzipielle Architektur des Systems. Der wesentliche Teil der Anfragelogik befindet sich in der zentralen DSMS-Komponente. Der INQP von *AnduIN* beschränkt sich auf die Ausführung von Anfragen. Ein detaillierter Überblick über die Aufgaben der einzelnen Komponenten folgt in Abschnitt 3.2.2 (INQP) und Abschnitt 3.2.3 (DSMS).

Im Folgenden soll zunächst auf den prinzipiellen Ablauf bei der Ausführung von Anfragen in diesem System eingegangen werden. Anschließend werden die einzelnen Komponenten und deren Aufgaben im Detail beschrieben.

3.2.1 Anfragen und Operatorgraphen

Im Abschnitt 2 wurden verschiedene Ansätze zur Verarbeitung von Datenströmen präsentiert. Um die Erweiterbarkeit des hier vorgestellten Systems möglichst einfach zu

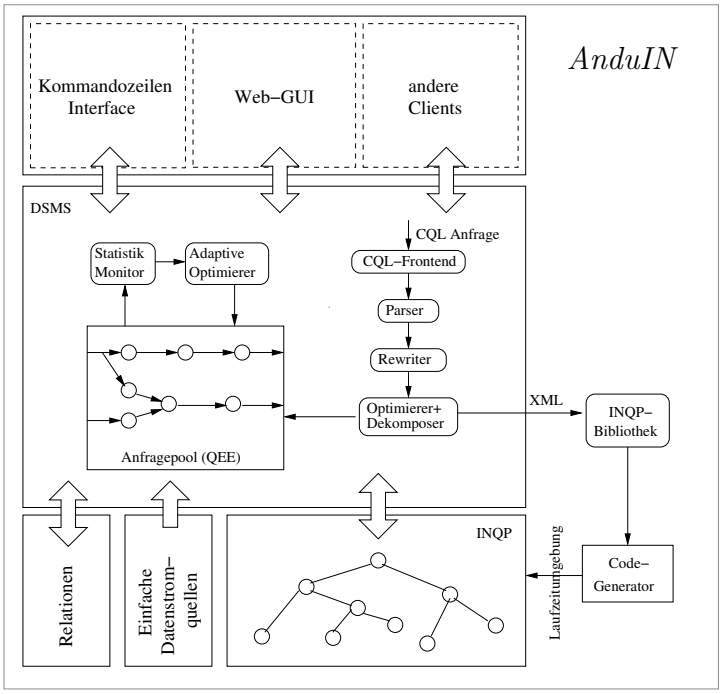


Abbildung 3.1: Grundlegende Architektur von *AnduIN*

gestalten, wird ein über beide Komponenten des System einheitlicher Ansatz gewählt. Zwar unterscheiden sich, bedingt durch die unterschiedlichen Ziel-Plattformen der Komponenten, die einzelnen Implementierungen der Algorithmen. Der prinzipielle Ablauf und die Einbindung der Operatoren ist allerdings gleich. Zudem erlaubt eine systemweit einheitliche Bearbeitung von Anfragen eine einfachere Anfrageoptimierung und Kostenmodell-Entwicklung.

Die physische Verarbeitung von Anfragen innerhalb von *AnduIN* erfolgt auf Basis von Operatorgraphen. Ein Operatorgraph ist ein zyklenfreier, gerichteter Graph von Operatoren, welcher von eingehenden Tupeln in Richtung Wurzel durchflossen wird (siehe auch [46, 133]). Operatoren kapseln dabei tupel-verarbeitende Algorithmen und folgen dem Quelle-Senke-Prinzip. Damit existieren die folgenden drei Formen von Operatoren:

- *Quellen*, welche ausschließlich Tupel produzieren,
- *Senken*, welche ausschließlich Tupel konsumieren und
- *innere Operatoren*, welche sowohl Tupel konsumieren als auch produzieren.

Üblicherweise wird eine Anfrage von einer Menge von Quellen (mindestens eine), einer Menge von inneren Operatoren und genau einer Senke repräsentiert (der Anfrage-Ausgabe beim Nutzer). Die Operatoren selbst sind mittels eines *publish/subscribe*-Mechanismus verbunden, so dass sich die Operatorgraphen auf einfache Weise zur Laufzeit anpassen lassen. Prinzipiell ist jeder Operator in der Lage, Tupel an beliebige

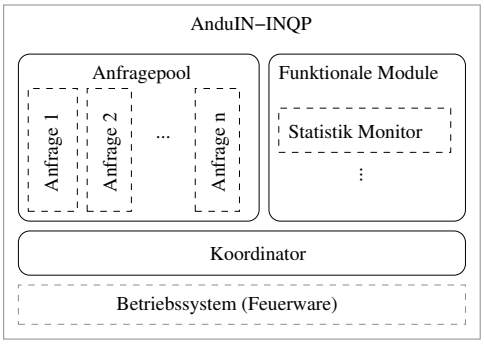


Abbildung 3.2: Modularer Aufbau des *AnduIN* INQP

Senken zu senden. Zum gegenwärtigen Zeitpunkt gilt dies in *AnduIN* aber lediglich für Quellen.

Aus Gründen der Einfachheit wird auf Tupelbuffer zwischen den einzelnen Operatoren im Operatorgraphen verzichtet. Sollte dies gewünscht sein, so können einfache Buffer-Operatoren, welche zusätzlich zwischen den Operatoren eingefügt werden, diese Aufgabe übernehmen. Hat ein innerer Operator die Verarbeitung eines Tupels abgeschlossen und wurde ein Ergebnis produziert, dann wird dieses umgehend an alle bei ihm registrierten Operatoren gesendet.

Um auf Verarbeitungs-Engpässe durch zu hohe Auslastung einzelner Operatoren reagieren zu können, sind die Quelloperatoren in der Lage, die Datenweitergabe an registrierte Operatoren zu pausieren. Dies geschieht so lange, bis wieder ausreichend Ressourcen für alle Operatoren zur Verfügung stehen.

3.2.2 In-Network Query-Prozessor

Aufgrund der Anforderung, nahezu beliebige Operatoren im Netzwerk anwenden zu können, ist der von TinyDB bekannte monolithische Ansatz nicht geeignet. Im Gegensatz dazu verwendet der INQP von *AnduIN* eine modulare Sensorknoten-Laufzeitumgebung. Hierbei wird ein ähnlicher Ansatz verfolgt, wie dies zum Beispiel bei PIPES [133] der Fall ist. PIPES stellt eine Bibliothek zur Verfügung, welche es ermöglicht, nahezu beliebige DSMS (in Abhängigkeit von der Applikation) zu entwickeln. Im Fall der In-Network-Komponente von *AnduIN* wird diese Idee weitergeführt. Die Laufzeitumgebung der Sensorknoten wird dabei abhängig von einer Menge konkreter Anfragen entwickelt und anschließend im WSN verteilt.

Der modulare Aufbau erlaubt die Integration prinzipiell beliebiger Funktionalität in die Lauzeitumgebung, sofern die auf den Sensorknoten zur Verfügung stehenden Ressourcen nicht überschritten werden. Der INQP-Teil von Anfragen bzw. die notwendigen, miteinander verknüpften Operatoren werden in den INQP-Anfragepool der Lauzeitumgebung integriert. Zum Verarbeitungszeitpunkt werden die Anfragen durch entsprechende Timer des Sensorknoten-Betriebssystems aktiviert.

Neben den eigentlichen Anfragen müssen weiterhin zusätzliche funktionale Module eingebunden werden. Diese dienen der Prozessüberwachung und sind u.a. für die Reoptimierung von Anfragen notwendig. Hierbei werden lediglich die benötigten Komponenten in die Laufzeitumgebung integriert (zum Beispiel ein Statistik-Monitor oder der Koordinator, welcher die Quell-Operatoren aktiviert). Sämtliche mit der Anfrageoptimierung verbundene Systemkomponenten sind Teil der zentralen Instanz. Der Verzicht auf diese Komponenten innerhalb des WSN verringert den Speicherverbrauch und erlaubt somit die Integration weiterer Anfragen bzw. Anfrageteile. Abbildung 3.2 zeigt schematisch den prinzipiellen Aufbau der Laufzeitumgebung der INQP-Komponente von *AnduIN*.

Ein weiterer Vorteil des modularen Aufbaus in Verbindung mit dem einfachen *publish/-subscribe*-Mechanismus ist die generelle Möglichkeit der Wiederverwendung von einzelnen Operatoren oder ganzen Anfrageteilen in anderen Anfragen. Auf die Möglichkeiten der Multi-Query-Optimierung (MQO) im Kontext von *AnduIN* soll in Abschnitt 4.5 eingegangen werden.

Sämtliche Operatoren und die für die Anfrage-Verarbeitung notwendigen Module der In-Network-Komponente von *AnduIN* liegen in Form einer C-Bibliothek auf der zentralen Instanz vor. Die einzelnen Module sind dabei als Komponenten des eigentlichen Sensorknoten-Betriebssystems implementiert.

Auf Basis des von der zentralen Komponente bestimmten INQP-Anfrageplanes erzeugt ein Code-Generator aus der Operator-Bibliothek und dem Sensorknoten-Betriebssystem die Laufzeitumgebung. Dieser kann anschließend (manuell oder automatisch) auf die Sensorknoten übertragen werden. Dabei ist zu beachten, dass für alle Knoten des WSN die gleiche Laufzeitumgebung generiert wird. Eine Spezialisierung der Sensorknoten ist ausschließlich auf Basis eines Knotenidentifikators vorgesehen. Die Laufzeitumgebung enthält somit die Module für alle Knoten im WSN. Diese können in Abhängigkeit der Sensorknotenkennung frei geschaltet werden.

Dem Wurzelknoten des WSN kommt eine besondere Bedeutung zu. Dieser muss die aus dem Netzwerk empfangenen Daten aufbereiten und der DSMS-Komponente von *AnduIN* als Eingabestrom zur Verfügung stellen. Die Definition der notwendigen Kommunikationsschnittstelle zum DSMS erfolgt dabei zum Zeitpunkt der Anfrageanalyse automatisch und ist Teil der Anfragespezifikation durch die zentrale Komponente.

3.2.3 Datenstrom-Managementsystem

Die Implementierung der zentralen Komponente von *AnduIN* basiert zu großen Teilen auf dem im Fachbereich Datenbanken und Implementierungstechniken der TU Ilmenau entwickelten DSMS *StreamDB* [110,153] und beinhaltet u.a. ein CQL-Interface, welches dem Linear Road Benchmark genügt [17]. Das System wurde dabei in nahezu allen Komponenten um die Möglichkeit der partiellen In-Network-Verarbeitung erweitert, wobei wesentliche Teile der Anfrageverarbeitung (Parser, Anfrage-Rewriter und Query Execution Engine), sowie die primitiven DSMS-Operatoren des StreamDB-Projektes in die Implementierung von *AnduIN* übernommen werden konnten. Andere Teile, wie

zum Beispiel die physische Anfrageoptimierung, mussten grundlegend neu entwickelt werden.

Vom System entgegengenommene Anfragen werden analysiert, optimiert und anschließend bei Bedarf für die weitere Verarbeitung partitioniert. Die Arbeitsweise der Anfrageanalyse entspricht dabei zum großen Teil der von traditionellen DBMS. Eingehende Anfragen werden vom *Frontend* (notwendig für den Multi-Client-Betrieb von *AnduIN*, siehe Abschnitt 6.1) entgegengenommen, vom *Parser* auf ihre syntaktische Korrektheit hin geprüft und anschließend in einen logischen Anfrageplan überführt. Der *Rewriter* übernimmt die Sichtexpansion und führt eine regelbasierte logische Optimierung durch. Anschließend übersetzt der physische *Optimierer* den logischen Anfrageplan in eine Menge von physischen Ausführungsplänen, aus denen der auszuführende Plan gewählt wird. Bei der Überführung der logischen Operatoren in ihre physischen Äquivalente hat der Optimierer die Möglichkeit, zwischen Operatoren zu wählen, welche ausschließlich lokal von der DSMS-Komponente ausgeführt werden und welche partiell oder vollständig vom INQP bearbeitet werden. Die Wahl des auszuführenden Planes – und damit verbunden die Wahl der Komponente, von welcher die Operatoren verarbeitet werden sollen – basiert auf einem multidimensionalen Kostenmodell. Kapitel 4 beschreibt die Anfrageoptimierung im Detail.

Enthält der auszuführende Anfrageplan Teile, welche vom INQP bearbeitet werden sollen, dann müssen zuerst die entsprechenden Operatoren identifiziert und anschließend aus dem Anfrageplan herausgelöst werden. Der physische Anfrageplan wird also in einen im DSMS und einen vom INQP auszuführenden Teil zerlegt. Der für den INQP bestimmte Teil des Anfrageplanes wird dabei in eine XML-Spezifikation überführt und anschließend an einen Codegenerator übergegeben. Dieser übernimmt das finale Erzeugen der Laufzeitumgebung für die Sensorknoten. Der vom DSMS zu bearbeitende Teil wird an den zentralen Anfragepool (*query execution engine*-(QEE)) weitergeleitet, welche sämtliche lokal auszuführenden Anfragen verwaltet.

Durch die XML-basierte Spezifikation des WSN-Teils von Anfragen können prinzipiell beliebige INQP in *AnduIN* [53] integriert werden. Hierzu müssen dem System lediglich die vom verwendeten INQP angebotenen Operatoren und deren Ausführungskosten bekannt gegeben werden. Darüber hinaus muss ein Codegenerator für die Übersetzung der XML-Spezifikation in den Code der Laufzeitumgebung bereitgestellt werden.

Da sich die Datenverteilungen an den Quellen während der Anfrageausführung ändern können, ist gegebenenfalls eine Adaption von Anfragen zur Laufzeit notwendig. Zu diesem Zweck werden laufende Anfragen der QEE kontinuierlich von einem *Statistik-Monitor* überwacht. Die von dem Monitor erfassten Werte können anschließend vom *Adaptiven Optimierer* zur Anpassung der laufenden Anfragen herangezogen werden.

Die DSMS-Komponente von *AnduIN* kann prinzipiell mit beliebigen Datenstrom- und Datenbankquellen umgehen, so dass eine Verknüpfung der vom WSN produzierten Daten mit zusätzlichen Daten gewährleistet ist. Die Anbindung von externen Datenströmen erfolgt dabei über einfache Netzwerksockets, die Integration von Datenbankquellen über ein entsprechendes Wrapper-Interface.

3.3 Deklarative Anfragebeschreibung

Während im Bereich der relationalen Datenbanksysteme mit SQL eine standardisierte Anfragesprache existiert, gibt es im Bereich der datenstrom-verarbeitenden Systeme gegenwärtig keinen einheitlichen Standard. Meist wird mit der Entwicklung eines neuen DSMS auch eine neue Anfragesprache präsentiert (zum Beispiel CQL [15], ACQL [149], WXQuery [136], SPADE [81] und SQL Extensions [138]).

Oftmals orientieren sich die präsentierten Lösungen zu wesentlichen Teilen am SQL-Standard. Die meisten Erweiterungen beziehen sich auf die Integration von Synopsenstrukturen zur Auflösung blockierender Operatoren (siehe Abschnitt 2.2.1) und auf Möglichkeiten zur Definition von Anfragen der verschiedenen Anfrageklassen (Snapshot-, Event- und kontinuierliche Anfragen). Im Kontext der datenstrom-verarbeitenden Systeme wird jedoch oftmals davon ausgegangen, dass es sich bei den zu beschreibenden Anfragen um kontinuierliche Anfragen handelt. Snapshot- bzw. Event-Anfragen finden sich eher im Bereich der INQP.

Aufgrund der Dominanz von TinyDB bei den INQP hat sich mit ACQL ein Quasi-Standard etabliert, welcher für weitere Arbeiten auf diesem Gebiet als Grundlage diente. Neben den bereits erwähnten typischen Erweiterungen von Datenstrom-Anfragesprachen integriert ACQL zusätzlich noch Konstrukte zur Datenerzeugung. Eine in ACQL formulierte Anfrage enthält somit zum Beispiel neben der Gültigkeitsdauer oder Konstrukten zum Auslösen von Ereignissen zusätzlich noch Informationen darüber, wann eine in der Anfrage spezifizierte Datenquelle Werte zu erfassen hat. Diese implizite Art der Datendefinition in Anfragen weist allerdings einige grundlegende Nachteile auf:

- Die Beschreibung der Datenerzeugung muss in jeder Anfrage neu vorgenommen werden, eine Wiederverwendung ist nicht vorgesehen.
- Anfragen werden durch die zusätzlichen Informationen komplexer und für den Anwender unübersichtlich. Komplexe, verschachtelte Anfragen lassen sich nur schwer oder gar nicht definieren.
- Das Konzept der impliziten Datendefinition ist nur eingeschränkt auf Nicht-WSN Daten anwendbar. Es existiert somit kein einheitliches Modell zur gleichzeitigen Verwendung von Daten aus WSNs, einfachen Datenströmen und statischen Relationen.

Im Folgenden soll der in *AnduIN* umgesetzte Ansatz zur Anfragedeklaration vorgestellt werden. Die Integration des WSN als eine mögliche Datenquelle unter vielen spiegelt sich dabei in der Umsetzung der Anfragesprache wider. Diese wurde in Anlehnung an das im Datenbankbereich übliche DDL-DML-Schema in einen Teil zur Definition von Datenquellen und einen Teil zur Definition von Anfragen aufgespaltet. Für den Nutzer sollte es transparent sein, ob es sich bei einer Datenquelle um ein WSN oder einen beliebigen Datenstrom handelt, er soll lediglich das zu erreichende Ziel spezifizieren. Im Datenstromkontext bedeutet dies die Trennung in einen Sprachteil zur Datendefinition

(DDL) und die eigentliche Anfragesprache, welche im Fall von *AnduIN* auf CQL basiert. Im Weiteren sollen Teile der DDL von *AnduIN* bzw. im Zusammenhang mit der Entwicklung von *AnduIN* eingeführte Erweiterungen von CQL beschrieben werden.

3.3.1 Data Definition Language

Unter dem Begriff der *Data Definition Language* (DDL) fasst man im Bereich der Datenbanksysteme eine Menge von Sprachkonstrukten zusammen, welche der Beschreibung von Datenstrukturen dienen (zum Beispiel Relationen, Attribute, Datentypen etc.).

Auf den Bereich der DSMS übertragen, bezieht sich die DDL also auf die Beschreibung von möglichen (kontinuierlichen und nicht-kontinuierlichen) Datenquellen. Je nach DSMS sind verschiedene Quellen für die weitere Anfrageverarbeitung vorgesehen. Das hier beschriebene System, *AnduIN*, unterstützt die folgenden Datenquellen:

- *Relationen* sind Mengen von Tupeln, welche sich über die Zeit nicht ändern. Die Daten liegen in einer statischen Datenbank vor.
- *Einfache Datenströme* sind Datenströme aus beliebigen Quellen. Dem System muss bekannt gegegeben werden, an welcher externen Systemschnittstelle die Daten zu erwarten sind und wie diese zu interpretieren sind (Attribute, Datentypen etc.).
- *Virtuelle Sensornetzwerke* sind eine im Rahmen von *AnduIN* eingeführte Klasse von Datenquellen. Sie bezeichnen eine virtuelle Datenstruktur über einer Menge von Sensorknoten, welche u.a. für eine effiziente In-Network-Verarbeitung von Daten herangezogen werden kann. Die Spezifikation von WSNs erfolgt in *AnduIN* in Form virtueller Sensornetzwerke.

Die Integration von Relationen und einfachen Datenströmen gestaltet sich ähnlich wie bei existierenden Lösungen. Relationen werden dabei mittels externer DBMS-Lösungen eingebunden. Zum Zeitpunkt der Anfrage-Initialisierung werden benötigte Relationen ausgelesen und in Analogie zum Relation-to-Stream-Operator von STREAM in einen Datenstrom verwandelt, welcher anschließend zur weiteren Verarbeitung an die entsprechenden Anfrageteile weitergeleitet wird.

Die Einbindung einfacher Datenströme erfolgt über die Netzwerkschnittstelle des zugrunde liegenden Betriebssystems der zentralen Komponenten. Ein entsprechender Listener hört auf einen bei der Registrierung des Datenstromes definierten Port und interpretiert eingehende Werte gemäß des spezifizierten Schemas. Details zum Anlegen von Relationen bzw. zum Registrieren von einfachen Datenströmen finden sich im Anhang C.

Für den Umgang mit WSN wurde in *AnduIN* das Konzept der virtuellen Sensornetzwerke als mögliche Datenstromquelle integriert. Im Weiteren soll hierauf im Detail eingegangen werden.

3.3.1.1 Virtuelle Sensornetzwerke

Ein virtuelles Sensornetzwerk ist ein Datenstrom, dessen Quellen Sensorknoten eines WSN sind. Das Erzeugen eines virtuelles Sensornetzwerkes hat dabei keinen Einfluss auf die Ausführung gegenwärtig von *AnduIN* verarbeiteter Anfragen. Es handelt sich lediglich um die Beschreibung einer möglichen Datenstrom-Quelle. Diese Beschreibung enthält zusätzliche Informationen zum Verhalten bei einer physischen Ausprägung des beschriebenen Netzwerkes. Erst die Verwendung eines so definierten Datenstromes in einer Anfrage hat u.U. Einfluss auf gegenwärtig auf dem Sensorknoten ausgeführte Operatoren. Die im Zusammenhang mit der Definition eines virtuellen Sensornetzwerkes spezifizierten Parameter kann *AnduIN* zur Analyse und Optimierung heranziehen.

Ein virtuelles Sensornetzwerk wird wie folgt definiert:

```
CREATE STREAM name (attribute type [, attribute type]) NETWORK
        [parameter value [, parameter value]]
```

Der Parameter *name* bezeichnet den Namen des Datenstromes, welcher von dem entsprechenden physischen Sensornetzwerk geliefert wird. Die Non-Terminale *attribute* und *type* definieren das Schema des gelieferten Stromes. Das Terminal `NETWORK` zeigt an, dass es sich bei dem so definierten Strom um eine Netzwerkquelle handelt. Mit den Parametern (*parameter*/`value`) lassen sich netzwerkspezifische Eigenschaften beschreiben (zum Beispiel das Samplingintervall, das zu verwendende Routingprotokoll, Informationen zu eventuell logischen Netzwerkschichten etc.).

Neben der strikten Trennung von Datenbeschreibung und Anfragen bietet der Ansatz der virtuellen Sensornetzwerke noch die folgenden weiteren Vorteile gegenüber anderen Ansätzen, welche netzwerkbeschreibende Elemente direkt in die Anfragebeschreibung integrieren:

- *einheitliche Definition von Datenstrom-Quellen*: In *AnduIN* wird eine einheitliche Sprache zur Datendefinition für beide Teile (DSMS und INQP), welche im System vereint sind, verwendet. Trotzdem wird auf die Besonderheiten der einzelnen Komponenten eingegangen.

- *unterschiedliche Konfiguration von Datenstrom-Quellen in einer Anfrage*: Der Zeitaufwand und die Energiekosten für das Auslesen physischer Sensoren können stark variieren, so dass es nicht in jedem Fall sinnvoll ist, für alle Sensoren gleiche Messraten zu definieren. Mit Hilfe der virtuellen Sensornetzwerke ist die Definition verschiedener Samplingintervalle auf einfache Art und Weise möglich. Die so definierten Datenströme können anschließend über einen Verbund einfach zusammengeführt werden.

Ein Beispiel für die Definition eines virtuellen Sensornetzwerkes könnte wie folgt aussehen:

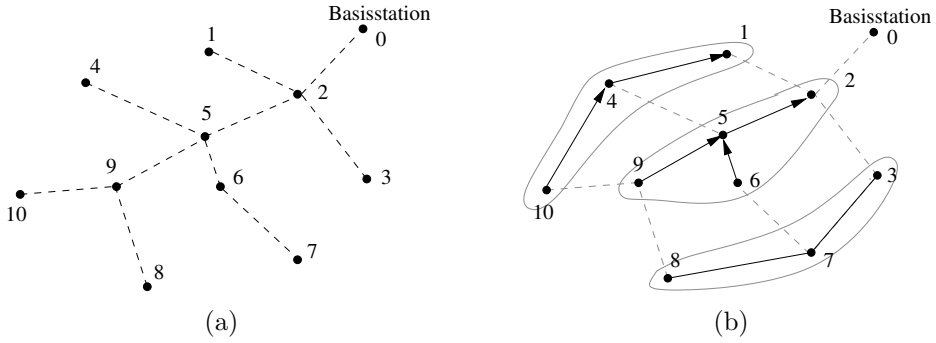


Abbildung 3.3: Autonomes Routing vs. manuelles Routing

```
CREATE STREAM net_stream (id int, temp double) NETWORK
        TOPOLOGY (15 (6, 9 )),
        SAMPLING 30 SECONDS
```

Hierbei wird ein Datenstrom *net_stream* registriert, welcher die beiden Attribute *id* und *temp* liefert. Im Fall einer Anfrage, welche das so definierte Netzwerk verwendet, erzeugen alle Sensorknoten, welche über die angefragten Attribute verfügen, Tupel mit dem definierten Schema. Sensoren, welche diese nicht produzieren können (zum Beispiel in einer heterogenen Systemumgebung), nehmen an der Ausführung der mit dem virtuellen Sensornetzwerk verknüpften Anfragen nicht teil.

Gemäß der Definition erfassen die Sensorknoten alle 30 Sekunden die angeforderten Messwerte. Zusätzlich wurde eine logische Netzwerk-Topologie über den Knoten mit den IDs 15, 6 und 9 definiert. Auf die Möglichkeit der manuellen Definition eines logischen Netzwerk-Overlays wird im nächsten Abschnitt eingegangen.

3.3.1.2 Logische Netzwerktopologien

Üblicherweise sind WSN selbstorganisierende Netzwerke, d.h. für die Kommunikation untereinander bauen die Knoten selbstständig Multi-Hop-Topologien auf, welche sich dynamisch an das Verhalten der Teilnehmer (zum Beispiel Knotenausfälle oder -bewegungen) anpassen. Die dafür notwendigen Mechanismen sind zumeist Teil des zugrundeliegenden Betriebssystems und Forschungsschwerpunkt verschiedener Projekte (zum Beispiel [105,111,176]). Neben diesen physischen Netzwerken wurden in der Literatur verschiedene Ansätze für das Bilden logischer Netzwerk-Overlays präsentiert. Ein Overlay strukturiert dabei ein Netzwerk unabhängig von dessen physischer Organisation, beispielsweise basierend auf in der Anfrage formuliertem semantischem Wissen, und wird vom System zumeist selbstständig aufgebaut.

Im Allgemeinen ist das automatische Erzeugen von Netzwerktopologien erstrebenswert. Dennoch gibt es (insbesondere im Kontext komplexer Anfragen und Operatoren) Bedarf, den Datenfluss innerhalb eines WSN manuell zu bestimmen. Ein einfaches

Beispiel soll dies verdeutlichen: In einem modernen Gebäude befinden sich mehrere hundert Sensoren. Anhand der von den Sensoren erfassten Daten sollen Regeln für die automatische Steuerung des Gebäudes abgeleitet werden. Ein möglicher Ansatz, solche Regeln abzuleiten, ist das Finden häufig auftretender Muster in den Datenströmen (siehe Abschnitt 5.3). Mit steigender Anzahl der zu analysierenden Datenströme steigt sowohl der Aufwand für die Analyse als auch der für eventuelle Synopsen notwendige Speicherbedarf. Da insbesondere im Zusammenhang mit drahtlosen Sensorknoten die Ressourcen stark begrenzt sind, wird eine Beschränkung der zu analysierenden Datenmenge notwendig. Ein möglicher Ansatz für das Frequent-Pattern-Mining-Problem wurde in [179] vorgestellt. Bei diesem werden die Muster über automatisch generierte logische Nachbarschaften hinweg ermittelt. In einem alternativen Ansatz ließe sich das vom Architekten vorhandene Wissen nutzen, um gemeinsam zu analysierende Regionen manuell zu definieren. So ist es zum Beispiel offensichtlich, dass Sensoren in einem Raum stärker miteinander in Beziehung stehen als Sensoren in räumlich weit voneinander getrennten Bereichen eines Gebäudes. Solche Eigenschaften können einem WSN nur durch Zuführung externen Wissens bekannt gegeben werden. Ist dies nicht möglich, so müsste beispielsweise im Fall des in Abbildung 3.3(a) dargestellten Netzwerkes eine Analyse über elf Datenströmen in Echtzeit erfolgen. Bei den in Abbildung 3.3(b) manuell definierten Gruppen sind jeweils nur drei bzw. vier Datenströme zu analysieren.

In Zusammenhang mit der Einführung des Konzepts virtueller Sensornetzwerke wurde die Möglichkeit der Entwicklung logischer Netzwerk-Overlays geschaffen. Diese ermöglichen die Zuführung von externem Expertenwissen in das System. Das Routing, basierend auf den logischen Overlays, verhält sich orthogonal zu den physischen Routingstrategien. Die Kommunikation zweier im logischen Overlay benachbarter Knoten erfolgt über die physische Netzwerkschicht und deren Routingtabellen. Im Beispiel in Abbildung 3.3(b) bedeutet dies, dass eine Nachricht von Knoten 10 zu Knoten 4 über die Knoten 9 und 5 erfolgen muss, da keine direkte Kommunikation der beiden Knoten möglich ist.

Neben dem Erzeugen einfacher hierarchischer Zusammenhänge sollte es mit Hilfe eines logischen Overlays auch möglich sein, teilautonome Topologien aufzubauen. D.h., dass zum Beispiel logische Cluster/Gruppen manuell definiert werden, die Knoten eines Clusters aber selbstständig entscheiden können, welcher Knoten der Gruppe beispielsweise für die Kommunikation mit übergeordneten Knoten verantwortlich ist.

Das Registrieren eines logischen Overlays in *AnduIN* erfolgt im Rahmen der Registrierung eines virtuellen Netzwerkes. Das Overlay wird dabei durch Angabe der Knoten in In-Order-Reihenfolge (für Mehrwegebäume) definiert, wobei die Verarbeitungsreihenfolge der Kindknoten ignoriert wird. Befinden sich mehrere Bäume in einem Overlay, so werden diese, einfach durch Leerzeichen getrennt, konkateniert. Hiermit lassen sich nahezu beliebig komplexe Overlays definieren. Das Beispiel aus Abbildung 3.3(b) lässt sich wie folgt beschreiben:

```
(1(4(10))) (2(5(6,9))) (3,7,8)
```

Im Zusammenhang mit der Entwicklung von *AnduIN* wurde eine grafische Nutzer-

schnittstelle entwickelt (siehe Abschnitt 6.1.2). Um das Anlegen logischer Overlays möglichst einfach zu gestalten, wurde in das Nutzer-Interface ein entsprechender grafischer Editor integriert. Mit diesem lassen sich Topologien (über bereits registrierte Sensoren) per Drag-And-Drop erzeugen. Anschließend wird das so erzeugte Overlay in die eben beschriebene Form überführt.

Wurde eine Anfrage basierend auf einer logischen Netzwerktopologie erstellt, so wird diese in der Laufzeitumgebung in Form einer zusätzlichen Routingtabelle für die Operatoren zur Verfügung gestellt. Komplexe Operatoren, welche knotenübergreifend im WSN agieren, können diese für ihre Verarbeitung einsetzen.

3.3.2 Anfragesprache

Das im Kontext von STREAM eingeführte CQL [13] bildet die Grundlage der Anfragesprache von *AnduIN*. Da es sich bei CQL um eine der ersten Anfragesprachen für datenstrom-verarbeitende Systeme handelt, weißt diese allerdings einige Einschränkungen auf. Zum Beispiel existiert bisher keine Möglichkeit zur Integration komplexer Data-Mining-Operatoren in den Anfrageprozess. Da diese Funktionalität für die vorliegende Arbeit essentiell ist, musste eine entsprechende Erweiterung geschaffen werden, welche im Folgenden beschrieben wird.

CQL verfügt über die Möglichkeit der Definition von Synopsen-Operatoren, wobei im Kontext von CQL unter einer Synopse eine Datenstruktur zum Zwischenspeichern von Datenströmen verstanden wird. Die Autoren beschreiben in [13] zwar auch eine Möglichkeit mittels des Synopsen-Operators Datenzusammenfassungen in die Anfragebeschreibung aufzunehmen, in STREAM selbst werden die Synopsen-Operatoren aber im Wesentlichen in Form von gleitenden Fenstern umgesetzt. Als reine Datenspeicher dienen sie hierbei der Auflösung blockierender Operatoren mit der Einschränkung, dass eine Synopse lediglich einem Operator als Zwischenspeicher dienen kann [16].

Betrachtet man komplexe Verfahren (zum Beispiel Clustering, Frequent Pattern Mining, Burst-Erkennung), so sind die verwendeten Synopsen zumeist direkt mit einer Menge von Algorithmen zur Verwaltung und Analyse verknüpft (siehe Abschnitte 5.2 und 5.3). Eine Verwendung der Datenstruktur in Kombination mit anderen Operatoren ist zumeist nicht vorgesehen bzw. nicht sinnvoll. Auf dieser Grundlage basierend, wurde das in CQL vorgestellte Synopsen-Konzept um die Möglichkeit der Integration komplexer Verfahren mit eingebetteten Synopsen erweitert.

Die meisten in der Literatur zu findenden Data-Mining-Verfahren analysieren eine Menge von gleichzeitig eintreffenden Daten und deren Veränderungen relativ zueinander. Hierauf basierend wurde das mit CQL eingeführte Konzept der Synopsen über Datenströmen so erweitert, dass komplexe Verfahren über allen Attributen eines Datenstromes definiert werden können. Genauer bedeutet dies, dass, anstatt lediglich die Eigenschaften einer Datenzusammenfassung zu beschreiben (zum Beispiel die Größe eines Zeitfensters), das mit der Synopse verbundene komplexe Verfahren im Detail beschrieben wird (dies beinhaltet oftmals auch Beschreibungen der eigentlichen Synopse –

Dimensionen, Größe etc.). Der erweiterte Synopsenoperator hat dabei folgende Syntax

```
SELECT * FROM mystream [ synopsis (params) ]
```

Der Parameter `synopsis` entspricht nun nicht mehr der einfachen Beschreibung einer Datenstrukur, sondern definiert ein konkretes Verfahren, dessen genaue Eigenschaften sich über *params* definieren lassen.

Mit obiger Syntax können auf einfache Weise beliebige komplexe Verfahren in CQL integriert werden. Die Analyse findet dabei auf allen im Schema von *mystream* enthaltenen Attributen, statt. Sollen einzelne Attribute von der Analyse ausgeschlossen werden, so kann dies durch eine vorausgestellte Projektion geschehen. Zusätzlich können mit Hilfe des `exclude`-Parameters gezielt Attribute von der Analyse ausgeschlossen werden, auch wenn diese im Strom selbst noch vorhanden sind. Dies ist zum Beispiel bei Identifiern oder Zeitstempeln sinnvoll bzw. notwendig.

Die Ausführung von Operatoren in *AnduIN* wird durch das Eintreffen neuer Tupel aktiviert. Hierbei werden prinzipiell zwei Verarbeitungsweisen unterstützt:

- *kontinuierliche Verarbeitung*: Beim Eintreffen eines neuen Tupels wird die Aktualisierung der Synopse im Speicher umgehend vorgenommen, gefolgt von der sofortigen Auswertung der aktualisierten Synopsenstruktur. Am Ende dieser Analyse steht die Produktion einer Menge von Ausgabetupeln, welche an alle abhängigen Operatoren weitergeleitet wird.

- *batchweise Verarbeitung*: Bei dieser Verarbeitungsart erfolgt die Auswertung zyklisch nach einem definierten Zeitintervall bzw. einer vorgegebenen Anzahl von eingehenden (aktivierenden) Tupeln. Bei der batchweisen Verarbeitung sind wiederum zwei Ansätze möglich: (i) eintreffende Tupel führen umgehend zu einer Aktualisierung der Synopse oder (ii) neue Tupel werden so lange gepuffert, bis das Ende eines Batches erreicht ist und die Verarbeitung gestartet wird. In beiden Fällen erfolgt die Analyse erst am Batchende. Als Ergebnis eines Batches wird wiederum eine Menge von Ausgabetupeln erzeugt. Zwischen zwei Analyseschritten am Batchende findet keine Ausgabe von Tupeln statt. Der Synopsen-Operator arbeitet in dieser Zeit als reine Datensenke.

Prinzipiell wäre noch ein *Ad-Hoc*-Ansatz denkbar, welcher aber nicht in *AnduIN* integriert wurde (die Arbeit beschränkt sich auf die Verarbeitung kontinuierlicher Anfragen). Der *Ad-Hoc*-Ansatz unterscheidet sich von den beiden anderen Verarbeitungsarten dadurch, dass beim Eintreffen eines neuen Tupels lediglich die Synopsenstruktur aktualisiert werden muss. Die Auswertung der Synopse muss explizit durch den Nutzer oder eine Anwendung initiiert werden. Bei diesem Ansatz hat der Anwender die Möglichkeit, die gewünschte Auswertung vorzunehmen (die dafür notwendigen Daten werden in komprimierter Form vorgehalten). Wann die Analyse aber letztendlich durchzuführen ist, wird vom Nutzer bestimmt.

Die Verarbeitung eines Tupels innerhalb eines Synopsen-Operators bezieht sich immer auf alle Attribute des Tupels. Je nach Implementierung wird über jedem Attribut eine

einzelne Synopse oder eine Synopse über alle Attribute hinweg erzeugt und verwaltet. Das Ausschließen von Attributen von der Verarbeitung durch den *exclude*-Parameter ist nur bei Synopsen-Operatoren mit kontinuierlicher Verarbeitung möglich. In diesem Fall wird der unveränderte Wert beibehalten und an der entsprechenden Stelle im produzierten Tupel ausgegeben (ähnlich den einfachen Aggregat-Operatoren). Bei der batchweisen Verarbeitung ist dies nicht möglich, da der Zusammenhang zwischen eingehenden Tupeln und den am Batchende erzeugten Tupeln nicht hergestellt werden kann.

Im Rahmen der vorliegenden Arbeit wurde eine Reihe von komplexen Operatoren als Synopsen-Operatoren umgesetzt. Eine vollständige Liste der Operatoren, deren Parameter sowie weitere von *AnduIN* unterstützte Sprach-Elemente finden sich in Anhang C.

3.4 Zusammenfassung

In Kapitel 2 wurden verschiedene offene Probleme im Bereich der In-Network-Verarbeitung in drahtlosen Sensornetzwerken identifiziert. Hierauf basierend wurden im zurückliegenden Kapitel Anforderungen beschrieben, denen ein System genügen muss, um die ermittelten Probleme zu lösen. Mit *AnduIN* wurde ein im Rahmen dieser Arbeit entwickeltes System präsentiert, welches den gestellten Anforderungen genügt (siehe Abschnitt 6.2). Neben einer Vorstellung der grundlegenden Architektur des System, wurde dabei insbesondere auf die Nutzer-Systemschnittstelle eingegangen (die deklarative Anfragebeschreibung). Hervorzuheben sind insbesondere die Erweiterungen, welche im Kontext des manuellen Deployments und der Integration komplexer Analyseverfahren notwendig wurden.

Das zurückliegende Kapitel liefert lediglich einen allgemeinen Überblick über das entwickelte System. Aufgrund der Komplexität der Anfrageverarbeitung soll im Folgenden Kapitel 4 gesondert auf diese und die damit verbundenen Probleme eingegangen werden.

Kapitel 4

Anfrageverarbeitung

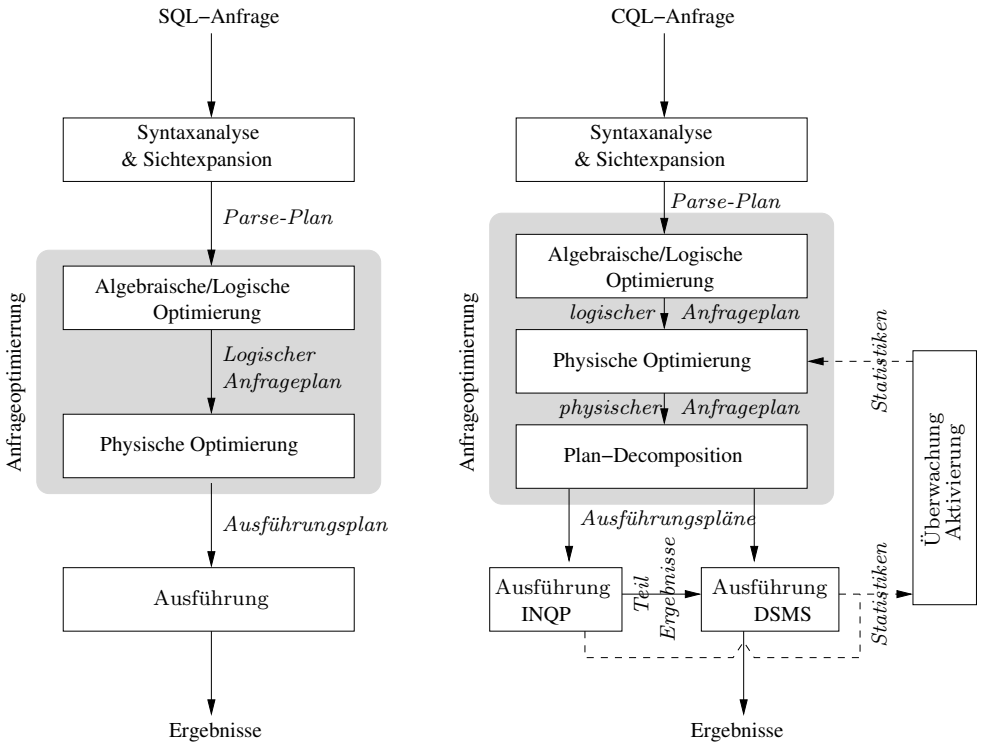
Unter dem Begriff der Anfrageverarbeitung versteht man die wesentlichen Schritte von der Spezifikation einer Aufgabe in Form einer Anfrage bis hin zu deren physischer Ausführung und eventuell notwendiger Adaptionen. Prinzipiell existieren zwei gegensätzliche Möglichkeiten der Anfrageformulierung. Beim *imperativen Ansatz* muss vom Anwender definiert werden, wie eine gegebene Aufgabe gelöst werden soll. Ein Beispiel aus der Datenbank-Welt sind die hierarchischen Netzwerkdatenbanken, bei welchen der Nutzer durch Navigationsbefehle beschreibt, wie er gewünschte Daten erreicht. Im Gegensatz hierzu steht der *deklarative Ansatz*, bei welchem sich der Anwender auf die Beschreibung der gewünschten Ergebnisse beschränkt. Wie diese Ergebnisse möglichst effizient erreicht werden, ist Aufgabe des verwendeten Systems. Heutzutage dominiert der deklarative Ansatz den Markt der Datenbanksysteme.

Das im Rahmen der Arbeit entwickelte System *AnduIN* folgt dem deklarativen Entwicklungsmodell. Die Entscheidungen, welcher Plan der beste ist und auf welcher Komponente des Systems dieser bearbeitet wird, wird automatisch vom System bestimmt. Die Anfrageverarbeitung folgt dabei den aus traditionellen DBMS bekannten Arbeitsschritten (siehe Abbildung 4.1(a) – auf die getrennte Darstellung von Laufzeitkomponenten (Parametrisierung, Code-Erzeugung) wurde dabei verzichtet):

1. *Syntaxanalyse & Sichtexpansion*: Die Anfrage wird auf ihre syntaktische Korrektheit hin überprüft. Anschließend wird eine Sichtexpansion durchgeführt, während die verwendeten Sichten aufgelöst werden. Das Ergebnis von Syntaxanalyse und Sichtexpansion ist ein der Anfrage entsprechender Parse-Plan.

2. *algebraische / logische Optimierung*: Zu Beginn der logischen Optimierung wird der Parse-Plan in einen initialen logischen Ausführungsplan übersetzt, welcher einer algebraischen Beschreibung der Anfrage entspricht. Der initiale logische Plan ist zu diesem Zeitpunkt noch nicht optimiert.

   Mit dem Ziel einer ersten Optimierung wird der initiale logische Ausführungsplan unter Verwendung algebraischer Transformationen in einen semantisch äquivalen-



(a) DBMS nach [181] (b) *AnduIN*

Abbildung 4.1: Anfragverarbeitung in DBMS und *AnduIN*

ten Plan übersetzt, dessen Ausführung üblicherweise effizienter ist als die des initialen Planes. Das Ergebnis ist ein optimierter logischer Anfrageplan.

3. *physische Optimierung*: Ziel der physischen Optimierung ist die Transformation des logischen Planes in einen Ausführungsplan unter Einbeziehung verschiedener Implementierungen der Operatoren. Im Allgemeinen ist die physische Optimierung ein mehrstufiger Prozess. In einem ersten Schritt werden während einer Enumerationsphase alle aus dem logischen Plan ableitbaren physischen Pläne ermittelt. Anschließend werden alle Pläne einer Bewertung (auf Basis eines Kostenmodells) unterzogen. Schließlich wird ein kostenminimaler Plan ausgewählt und an die Query Execution Engine weitergeleitet, welche für dessen Ausführung sorgt.

Logische Optimierung und *physische Optimierung* werden auch unter dem Begriff der Anfrageoptimierung zusammengefasst. Üblicherweise ist dabei das Ziel, einen Plan zu ermitteln, dessen Ausführung voraussichtlich minimale Ausführungskosten verursacht. In traditionellen DBMS werden hierzu oftmals Seitenzugriffe und CPU-Kosten herangezogen.

Die Anfrageverarbeitung in einem DSMS ähnelt im Wesentlichen der innerhalb eines

DBMS. Es gibt jedoch einen elementaren Unterschied. Dieser ergibt sich aus der Form der verarbeiteten Anfragetypen. Bei DBMS-Anfragen handelt es sich zumeist um kurzzeitige Anfragen, deren Ende bereits zum Zeitpunkt der Initialisierung absehbar ist (eine Ausnahme hiervon bilden OLAP-Anfragen). Daher findet die notwendige Optimierung üblicherweise vor der eigentlichen Ausführung statt (*plan-first-execute-next*-Ansatz). Im Gegensatz dazu handelt es sich bei DSMS-Anfragen um lange laufende Anfragen, während deren Ausführung sich sowohl Datencharakteristika als auch Systemeigenschaften wie zum Beispiel CPU-Last oder Speicherauslastung verändern können. Diese veränderten Eigenschaften können Einfluss auf die Effizienz der Verarbeitung laufender Anfragen haben und machen daher eventuell eine Reoptimierung zur Laufzeit notwendig. Die meisten der in Abschnitt 2 vorgestellten Systeme (sowohl INQP als DSMS) verfügen daher über entsprechende Monitore und Optimierer, die für die notwendigen Anpassungen sorgen. Man spricht in diesem Zusammenhang auch von *adaptiver Anfrageoptimierung*.

Das Ergebnis der physischen Optimierung in einem traditionellen DBMS ist ein Ausführungsplan, welcher an die entsprechende QEE weitergeleitet und von dieser ausgeführt wird. In *AnduIN* trennt der physische Operatorplan noch nicht zwischen Operatoren, welche vom DSMS verarbeitet werden und solchen, die vom INQP ausgeführt werden. Diese Trennung erfolgt erst in einer anschließenden Dekompostitionsphase. Während dieser werden alle im WSN verarbeiteten Operatoren aus dem Anfrageplan herausgelöst und notwendige Operatoren für die Kommunikation zwischen beiden Systemen, also DSMS und INQP, eingefügt. Die *beiden* resultierenden Teilpläne werden anschließend von den jeweiligen Komponenten ausgeführt.

Im Folgenden werden die einzelnen Schritte der Anfrageverarbeitung in *AnduIN* im Detail betrachtet. In einem ersten Schritt werden für die weitere Betrachtung notwendige Begriffe in Abschnitt 4.1 geklärt. Anschließend wird in Abschnitt 4.2 das in *AnduIN* verwendete Kostenmodell für die Bewertungsphase vorgestellt.

In Abschnitt 4.3 werden notwendige Schritte für eine initiale Anfrageoptimierung betrachtet. Schwerpunkte sind hierbei die Integration komplexer (komponenten-übergreifender) Operatoren im Kontext der Planenumeration und die multi-kriterielle Bewertung und Optimierung der zuvor bestimmten physischen Pläne.

Zusätzlich zur initialen Optimierung ist in *AnduIN* eine Reoptimierung laufender Anfragen vorgesehen. Entsprechende Lösungsansätze sollen in Abschnitt 4.4 betrachtet werden, wobei insbesondere auf die Problemen mit dem in *AnduIN* verwendeten INQP eingegangen wird.

4.1 Vorbetrachtungen

Wie in Abschnitt 3.2.1 beschrieben, werden sowohl logische als auch physische Anfragepläne in *AnduIN* in Form von Operatorgraphen dargestellt. Operatoren entsprechen dabei der Abstraktion eines Verfahrens innerhalb des verarbeitenden Systems [54]. Prinzipiell handelt es sich bei Operator-Graphen um zyklenfreie, gerichtete Graphen. Im

Kontext von *AnduIN* ist der Begriff des Operator-Graphen auf einen Mehrwege-Baum beschränkt. Die Knoten des Baumes entsprechen dabei den einzelnen Operatoren und Kanten beschreiben den Datenfluss zwischen diesen.

Ein logischer Anfrageplan $q = \{\mathbf{O}, \mathbf{S}, \mathbf{z}\}$ besteht aus einer Menge von logischen Operatoren $\mathbf{O}$, einer Menge von Quellen $\mathbf{S} \subset \mathbf{O}$ und einer Datensenke $\mathbf{z} \in \mathbf{O}$. Ein *logischer Operator* bezeichnet dabei die Abbildungen der entsprechenden algebraischen Operatoren. Ein Beispiel hierfür ist der Verbund-Operator (*join*).

Üblicherweise stehen für jeden logischen Operator verschiedene physische Implementierungen (physische Operatoren) zur Verfügung. Jeder logische Operator $op_i \in \mathbf{O}$ kann daher in eine Menge von physische Operatoren $\{\bar{op}_i^1, ..., \bar{op}_i^k\} \subset \bar{\mathbf{O}}_i$ überführt werden. $\bar{\mathbf{O}}_i$ bezeichnet dabei die Menge aller vom System zur Verfügung gestellten physischen Operatoren. So kann zum Beispiel der Verbund-Operator als *nested loop join* aber auch als *hash join* implementiert werden.

Aufgrund der verschiedenen physischen Implementierungen eines logischen Operators erhält man zu einem logischen Anfrageplan q typischerweise eine Menge von semantisch äquivalenten physischen Ausführensplänen $\{\bar{q}_1, \bar{q}_2, ...\} = \bar{\mathbf{Q}}$. Das Ermitteln des kostengünstigsten Planes ist das Ziel der physischen Anfrageoptimierung.

Im Fall des in dieser Arbeit betrachteten kombinierten Systems vergrößert sich die Menge der verfügbaren physischen Operatoren zusätzlich. Bereits ein einfacher Operator wie zum Beispiel ein Filter kann sowohl auf der zentralen Komponente als auch im WSN auf den Sensorknoten bearbeitet werden. Entsprechend müssen die physischen Operatoren bezüglich ihrer Kosten im physischen Ausführungsplan getrennt betrachtet werden. Im Folgenden wird die Menge der zum Operator op_i gehörenden physischen Operatoren, welche auf der lokalen Komponente bereitgestellt werden, als $\bar{\mathbf{O}}_i^{\mathcal{L}}$ bezeichnet und die Menge der physischen Operatoren, welche im WSN zu realisieren sind, als $\bar{\mathbf{O}}_i^{\mathcal{N}}$ bezeichnet. Die Vereinigung $\bar{\mathbf{O}}_i = \bar{\mathbf{O}}_i^{\mathcal{N}} \cup \bar{\mathbf{O}}_i^{\mathcal{L}}$ beider Mengen entspricht der Menge verfügbarer Operatoren zu dem logischen Operator op_i.

Komplexe Operatoren

Die Abbildung komplexer logischer Operatoren auf eines ihrer physischen Äquivalente muss gesondert betrachtet werden. Aufgrund ihrer Komplexität ist es bei diesen Operatoren oft nicht möglich bzw. nicht nötig, die vollständige Verarbeitung im WSN durchzuführen.

Um dennoch eine einfache Kostenabschätzung vornehmen zu können, werden komplexe Operatoren in eine Menge von einfachen Operatoren (genauer in Operatorgraphen) aufgeteilt. Die resultierenden Operatoren sind dabei jeweils einer der beiden Komponenten INQP oder DSMS zugeordnet. Die Aufspaltung der Operatoren über die verschiedenen Komponenten hinweg erfordert das zusätzliche Einfügen von Operatoren für die Kommunikation zwischen den Komponenten. Diese Kommunikationsoperatoren müssen sowohl im Kostenmodell berücksichtigt werden als auch in Form entsprechender Routinen umgesetzt und in den Ausführungsplan integriert werden.

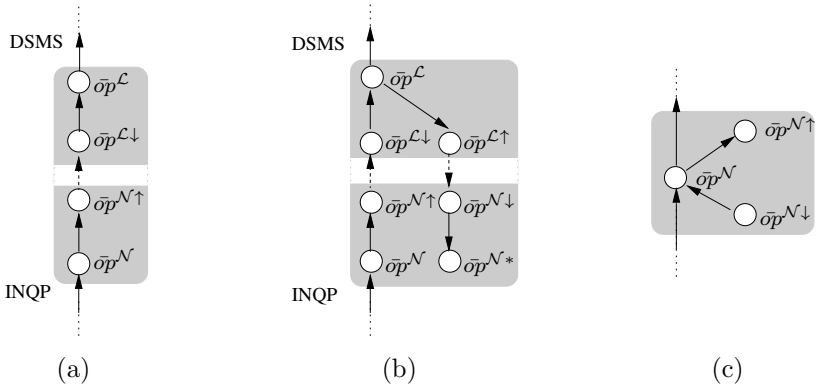


Abbildung 4.2: Abbildungen logischer komplexer Operatoren auf Mengen physischer Operatoren: (a) teilweise Verarbeitung im WSN, (b) feedback-basierte teilweise Verarbeitung im WSN, (c) Verarbeitung im WSN mit Datenaustausch zwischen den Knoten

Bei der Übersetzung komplexer logischer Operatoren können die nachstehenden Typen unterschieden werden.

Vollständige Verarbeitung im WSN Die Verarbeitung des Operators findet vollständig auf den Sensorknoten statt. Neben der Kommunikation mit seinen Quellen und der bei ihm registrierten Senke ist kein weiterer Informationsaustausch notwendig.

Teilweise Verarbeitung im WSN Die Verarbeitung des Operators wird in zwei verarbeitende Teile geteilt, einen auf der lokalen Instanz verarbeiteten Teil und einen von den Sensoren verarbeiteten Teil. Zusätzlich zu jedem Teil müssen noch entsprechende physische Operatoren zum Datenaustausch zwischen den beiden Lokalitäten eingefügt werden.

Abbildung 4.2(a) zeigt schematisch die Aufteilung eines teilweise im WSN verarbeiteten Operators. Die Kommunikation zwischen INQP und DSMS erfolgt über die beiden Operatoren $\bar{o}p^{\mathcal{N}\uparrow}$ (Senke) und $\bar{o}p^{\mathcal{L}\downarrow}$ (Quelle). Die Operatoren $\bar{o}p^{\mathcal{N}}$ und $\bar{o}p^{\mathcal{L}}$ repräsentieren die verarbeitenden Teile der physischen Operatoren. Durchgehende Pfeile beschreiben den Datenfluss zwischen Operatoren innerhalb der jeweiligen Komponente und eine gestrichelte Linie entspricht dem (Over-The-Air) Datentransfer zwischen den Komponenten. Ein gestrichelter Pfeil beschreibt die Kommunikation mit vorhergehenden bzw. nachfolgenden Operatoren im Ausführungsplan auf der jeweiligen Komponente. Die Kommunikation zwischen INQP und DSMS kann dabei in einem beliebigen Format stattfinden. Lediglich die beiden verarbeitenden Operatoren $\bar{o}p^{\mathcal{N}}$ und $\bar{o}p^{\mathcal{L}}$ müssen die ausgehenden Daten interpretieren können. Wichtig ist jedoch, dass sich die voraussichtlich ausgetauschte Datenmenge für die Kostenbewertung ermitteln lässt. Die Knoten $\bar{o}p^{\mathcal{N}\uparrow}$ und $\bar{o}p^{\mathcal{L}\downarrow}$ dienen lediglich der Kommunikation und haben keinerlei analytische Aufgabe.

Feedback-basierte teilweise Verarbeitung im WSN Die Verarbeitung des Operators erfolgt analog zur teilweisen Verarbeitung im WSN. Allerdings kann der auf der zentralen Instanz ausgeführte Teil des Operators Anpassungen an den Parametern des In-Network-Teils vornehmen (ähnlich einer Feedback Schleife). Die von der zentralen Instanz ausgehende Kommunikation ist wiederum mit zusätzlichen Kosten für das Senden/Empfangen der Daten im WSN verbunden. Dieser Typ der Zerlegung ist zum Beispiel bei dem in Abschnitt 5.3 beschriebenen Verfahren von Interesse. Die zentrale Instanz passt dabei Intervallgrenzen an, welche in das WSN propagiert werden müssen. Ein anderes Einsatzgebiet wären teilweise im WSN verarbeitete Operatoren, bei denen Eigenschaften aufgrund von Veränderungen, zum Beispiel der Auslastung, auf der zentralen Instanz angepasst werden müssen.

Abbildung 4.2(b) zeigt schematisch die Zerlegung dieses Operatortyps. Änderungen am INQP-Teil des komplexen Operators werden hierbei vom In-Network-Operator $\bar{o}p^{\mathcal{N}*}$ durchgeführt. Dieser Teil des komplexen Operators wird nur beim Eintreffen von Updates durch die zentrale Instanz aktiviert. Der Operator ist mit dem verarbeitenden In-Network-Teil $\bar{o}p^{\mathcal{N}}$ lediglich über das Anpassen von Parametern verbunden. Es erfolgt keine Aktivierung des Operators $\bar{o}p^{\mathcal{N}}$ durch $\bar{o}p^{\mathcal{N}*}$ wie es zum Beispiel beim Weiterleiten eines Tupels der Fall wäre. Die Operatoren $\bar{o}p^{\mathcal{N}\uparrow}, \bar{o}p^{\mathcal{N}\downarrow}, \bar{o}p^{\mathcal{L}\downarrow}, \bar{o}p^{\mathcal{L}\uparrow}$ repräsentieren wiederum die notwendigen Kommunikationsoperatoren.

Verarbeitung im WSN mit Datenaustausch zwischen den Sensorknoten Diese Art der Zerlegung weist den höchsten Grad an Verteilung auf. Der Operator wird hierbei komplett im Netzwerk verarbeitet. Für die Berechnung von Ergebnissen benötigt ein Operator Informationen von benachbarten Sensorknoten. Gleichzeitig muss er eigene Zwischenergebnisse den benachbarten Knoten für deren Berechnungen zur Verfügung stellen. Die zusätzliche Kommunikation wird ebenfalls in entsprechende Senken- und Quell-Operatoren ausgelagert. Diese sind für das Empfangen von Zwischenergebnissen benachbarter Knoten bzw. das Propagieren eigener Zwischenergebnisse verantwortlich.

Abbildung 4.2(c) zeigt die schematische Zerlegung eines Operators dieser Klasse. Der Operator $\bar{o}p^{\mathcal{N}}$ entspricht dabei der verarbeitenden Komponente. Die Operatoren $\bar{o}p^{\mathcal{N}\uparrow}$ bzw. $\bar{o}p^{\mathcal{N}\downarrow}$ sind für die Kommunikation zwischen den Sensorknoten im WSN verantwortlich. Prinzipiell ist es möglich, dass die Verarbeitung eines komplexen Operators dieser Klasse vollständig im WSN stattfindet. Der Operator muss nicht zwangsläufig über einen Teil auf der zentralen Instanz verfügen. Es ist auch ohne weiteres möglich, mehrere Operatoren dieser Klasse in einem einzigen Anfrageplan miteinander zu kombinieren. Die Kommunikationsschnittstelle kann dabei als Teil einer Netzwerkschicht angesehen werden, welche ausschließlich der Kommunikation dieses speziellen komplexen Operators dient.

In welche Klasse ein komplexer Operator fällt, muss je nach Implementierung entschieden und dem System mitgeteilt werden. Mischformen der genannten Typen sind dabei möglich. So ist es denkbar, dass ein im WSN ausgeführter physischer Operator sowohl von der zentralen Komponente Updates erfährt als auch mit seinen Nachbarknoten Zwischenergebnisse austauschen muss. Aufgrund des Einfügens zusätzlicher Senken (im Fall

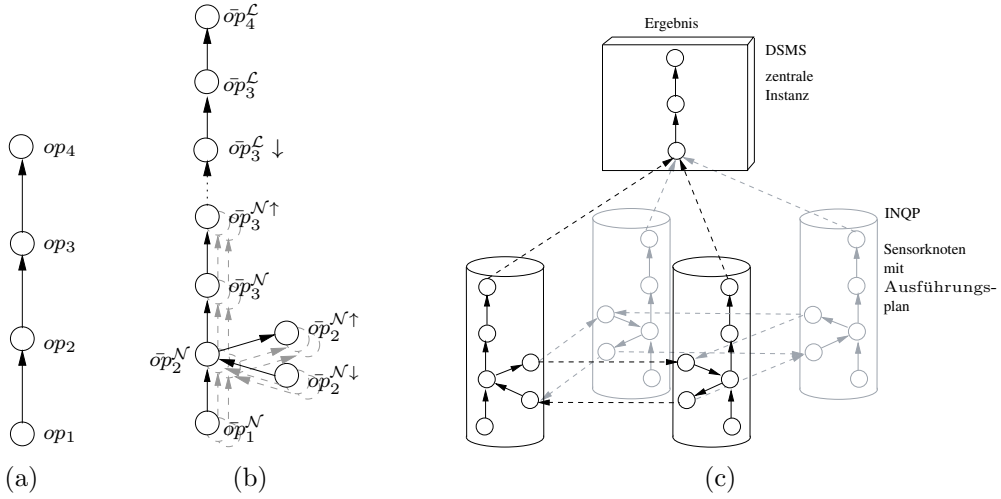


Abbildung 4.3: Beispieltransformation eines Anfrageplanes mit komplexen Operatoren: (a) logischer Anfrageplan, (b) physischer Ausführungsplan, (c) Ausführungsplan auf den jeweiligen Komponenten und deren Kommunikation

von Feedback-Operatoren) handelt es sich bei den durch die Transformation erzeugten physischen Operatorplänen nicht mehr um Mehrwege-Bäume.

Abbildung 4.3 zeigt exemplarisch die Transformation eines logischen Anfrageplanes in einen möglichen physischen Ausführungsplan. Der beim Ausführungsplan graue Anfrageteil entspricht dem von den Sensorknoten zu bearbeitenden Anteil der Operatoren. Die Vervielfältigung des INQP-Teils des Ausführungsplanes entspricht der Duplizierung auf die Sensorknoten (jeder Sensorknoten erhält exakt denselben Teil des Ausführungsplanes). In Abbildung 4.3(c) sind die im WSN verteilten Anfrageteile und der auf der zentralen Instanz ausgeführte Anfrageplan zu sehen. Die Kommunikation der Sensorknoten untereinander findet aus Gründen der Übersichtlichkeit in einem einfachen Ring statt. Prinzipiell sind hier auch komplexere Kommunikationspfade inklusive Multi-Hop-Routing möglich.

4.2 Kostenmodell

Bei traditionellen DBMS steht zumeist die Minimierung der Ausführungszeit im Vordergrund der Anfrageoptimierung. Die Ausführungskosten werden dabei durch verschiedene Parameter bestimmt, welche sich in zwei Kategorien gliedern lassen: (i) physische Parameter wie zum Beispiel die CPU-Kosten oder die I/O-Kosten für die Ausführung eines Operators und (ii) statistischer Parameter wie zum Beispiel Kardinalitäten von Relationen oder die Verteilung der Daten. Physische Parameter können durch ein einmaliges Messen bestimmt werden, statistische Werte hingegen müssen zur Systemlaufzeit erfasst werden.

Im Kontext der In-Network-Verarbeitung ist eines der wesentlichen Optimierungsziele die Lebenszeit des WSN. Die voraussichtliche Lebenszeit eines WSN wird dabei durch die vorhandenen (Energie-)Ressourcen und den Umgang mit diesen bestimmt. Ebenso wie bei der Kostenabschätzung in DBMS werden die Kosten dabei wiederum durch physische und statistische Parameter bestimmt. Im folgenden Abschnitt 4.2.1 soll ein Kostenmodell entwickelt werden, dessen Ziel die Approximation entstehender Energiekosten von Operator-Graphen ist.

Wie bereits erwähnt, ist eine ausschließliche Betrachtung der Energiekosten für die Anfrageoptimierung oftmals nicht ausreichend. Betrachtet man ausschließlich den Energieverbrauch, dann wäre die Deaktivierung aller entsprechenden Verbraucher die effizienteste Lösung. Dass dies nicht im Interesse des Anwenders liegen kann, ist offensichtlich. Daher sollte neben dem Energieverbrauch zumindest immer noch die Datenqualität als ein weiteres Optimierungsziel betrachtet werden. Das Problem des *Quality of Service* wurde bereits in einer Vielzahl von Untersuchungen (zum Beispiel [23, 32, 45, 207]) betrachtet. Bei manchen DSMS wie zum Beispiel Aurora [46] oder QStream [185, 186] war QoS sogar das entscheidende Entwicklungsziel. Aufgrund dieser Vielzahl von Arbeiten im Bereich QoS soll im Weiteren nicht darauf eingegangen werden. Der Einfachheit halber wird im vorliegenden Fall immer von vollständigen und korrekten Ergebnissen ausgegangen.

Ein weiteres, nicht minder interessantes Problem ist die Optimierung des Datendurchsatzes im DSMS. Im Kontext von *AnduIN* wird dieses Problem, insbesondere die Möglichkeit zur Integration weiterer WSNs oder zusätzlicher regulärer Datenstromquellen zur parallelen bzw. integrierten Auswertung, präsent. So ist es bei *AnduIN* ohne Weiteres möglich, vom WSN gelieferte Daten mit solchen aus externen Datenquellen zu verbinden, welche eventuell zuvor ebenfalls analysiert werden mussten. Das Problem der Echtzeitanalyse von Daten wird durch die Integration komplexer Operatoren noch zusätzlich verstärkt. Diese können im Allgemeinen deutlich weniger Datensätze pro Zeiteinheit verarbeiten, als dies zum Beispiel bei primitiven Operatoren der Fall ist. Um dennoch eine möglichst hohe Anzahl von parallelen Anfragen zu unterstützen, ohne dass es zu Engpässen innerhalb des Systems kommt, soll die Minimierung des Zeitaufwandes pro Anfrage im Weiteren als zweite Optimierungsdimension untersucht werden. Eine Abschätzung der pro Anfrage entstehenden Verarbeitungskosten wird in Abschnitt 4.2.2 betrachtet.

Aufgrund der Berücksichtigung von mindestens zwei Optimierungszielen handelt es sich bei dem vom System zu lösenden Optimierungsproblem um ein multi-kriterielles Problem. Prinzipiell lassen sich auch weitere notwendige und interessante Kriterien finden, welche in der Anfrageoptimierung berücksichtigt werden könnten. Da sich durch die Hinzunahme weiterer Dimensionen das zugrundeliegende Optimierungsverfahren nicht ändert, beschränkt sich die weitere Betrachtung auf die beiden Kriterien Energieverbrauch (Abschnitt 4.2.1) und Datendurchsatz (Abschnitt 4.2.2).

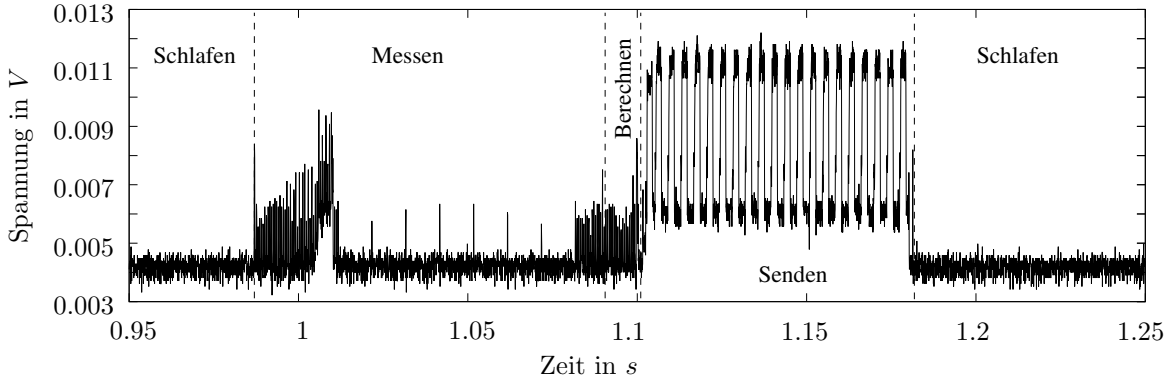


Abbildung 4.4: Phasen der Sensor-Verarbeitung

4.2.1 Energiekosten im WSN

Ziel des im Folgenden beschriebenen Kostenmodells ist es nicht, die realen Energieverbräuche zu ermitteln. Das hier präsentierte Kostenmodell dient lediglich dem Kostenvergleich verschiedener Pläne. D.h., ist die Ausführung einer Teil-Anfrage $\bar{q}_i$ auf den Sensorknoten mit einem höheren Energieverbrauch verbunden als die Ausführung einer Teil-Anfrage $\bar{q}_j$, so sollten die berechneten Kosten für $\bar{q}_i$ ebenfalls höher sein als die für $\bar{q}_j$.

Typischerweise werden beim Energieverbrauch eines Sensorknotens die folgenden Phasen unterschieden: (i) Mess-Phase, (ii) Verarbeitungsphase, (iii) Sende-Empfangsphase (Radio-Phase) und (iv) Schlaf-Phase. Abbildung 4.4 zeigt die verschiedenen Phasen während einer exemplarischen Energiemessung. Der Sensor erfasst dabei die Luftfeuchtigkeit und versendet anschließend sein Ergebnis an benachbarte Knoten.

Die aus energetischer Sicht dominierenden Phasen sind hierbei die Mess-Phase, während der Daten durch eventuell auf den Sensorknoten vorhandener physischer Sensoren erfasst werden, und die Radio-Phase, während der Nachrichten zu benachbarten Knoten gesendet oder von diesen empfangen werden. Aufgrund ihrer Dominanz wurden in bisherigen Kostenmodellen zumeist ausschließlich die Kosten für das Versenden und Empfangen von Daten für die Energieberechnungen herangezogen [149, 218].

Mit zunehmender Komplexität der Verarbeitung auf den Sensorknoten rücken allerdings auch die Kosten für die eigentliche Berechnung immer mehr in den Vordergrund. Es ist leicht einzusehen, das für einen einfachen Filter oder eine Projektion auf einem Sensorknoten keine wesentlichen Verarbeitungskosten anfallen. Muss aber zum Beispiel in jedem Zeitschritt eine komplexe Synopse aktualisiert werden oder sind die Daten aufgrund ihrer Größe teilweise auf einem Sekundärspeicher (zum Beispiel Flash) ausgelagert, dann fallen schnell Kosten in Dimensionen an, welche nicht mehr vernachlässigbar sind. Aus diesem Grund sollen die Berechnungkosten in das zu entwickelnde Kostenmodell mit einfließen.

Während vollständiger Inaktivität wird ein Sensorknoten üblicherweise vom Betriebssystem in den Sleep Mode (Schlaf-Phase) versetzt. Während dieser Phase werden al-

le nicht notwendigen Bauteile deaktiviert. In Abhängigkeit der eingesetzten Hardware existieren verschiedene Phasen dieses Zustandes (je höher das Sleep-Level, je mehr Bauteile werden deaktiviert). Allen Sleep-Modes gemein ist, dass sie aus energetischer Sicht effizienter sind als Verarbeitungsphasen.

Der Energieverbrauch eines Anfrageplanes hängt von zwei wesentlichen Aspekten ab: (i) der Aktivierungshäufigkeit der einzelnen Operatoren und (ii) der für eine einzelne Ausführung notwendigen Energie. Im Folgenden sollen beide Kriterien im Detail betrachtet werden.

4.2.1.1 Aktivierungshäufigkeit

Die Aktivierungshäufigkeit eines Operators $\bar{op}$, also die Anzahl an Ausführungen des Operators, entspricht der Summe der Ausgabedatensätze, welche von den Operatoren stammen, bei denen $\bar{op}$ registriert ist. Hierzu sollen zunächst die Begriffe Input-Rate, Output-Rate und Selektivität eines Operators erklärt werden.

Die Selektivität eines Operators ist als das Verhältnis eingehender und ausgehender Datensätze im Zeitintervall Δt bestimmt, d.h.

$$\sigma(\bar{op}) = \frac{out(\bar{op})}{in(\bar{op})}$$

wobei $in(\bar{op})$ der Anzahl der eingehenden Tupel am Operator $\bar{op}$ im Zeitintervall Δt entspricht und $out(\bar{op})$ die Anzahl der ausgehenden Tupel des Operators im gleichen Zeitintervall. Projektionen haben eine Selektivität von 1, Filter weisen eine Selektivität ≤ 1 auf. Verbund-Operatoren können auch zusätzliche Tupel produzieren, so dass deren Selektivität ≥ 1 sein kann.

Sowohl die Ankunftsrate eines Operators als auch dessen Outputrate können zur Laufzeit statistisch erfasst werden. Quellen und Senken weisen keine Selektivität auf, da diese ausschließlich Tupel produzieren bzw. konsumieren. Die Berücksichtigung von Operator-Selektivitäten im Zusammenhang mit kontinuierlichen, prinzipiell unendlichen Datenströmen wurde u.a. bereits in [45, 207, 215] ausführlich diskutiert.

Selektivität im WSN Obige Selektivitätsdefinition ist in dieser Form nur für auf der zentralen Instanz verarbeitete Operatoren sinnvoll. Im Fall einer Berechnung im WSN findet auf jedem Sensorknoten ein Teil der Verarbeitung des Operators statt. In einem WSN mit N Teilnehmern können verschiedene Datencharakteristika auf den Sensorknoten auftreten und somit auch N verschiedene Selektivititäten. Es ist leicht zu sehen, dass eine Mittelwertberechnung über die Selektivitäten der einzelnen Knoten keine gute Abschätzung für die Selektivität des Gesamtoperators liefert.

Im Folgenden wird ein In-Network-Operator als ein partitionierter Operator betrachtet, dessen Berechnung verteilt auf den Knoten des WSN stattfindet, d.h. $\bar{op}^{\mathcal{N}} = \{\bar{op}_1^{\mathcal{N}}, \bar{op}_2^{\mathcal{N}}, \ldots, \bar{op}_N^{\mathcal{N}}\}$. Die Anzahl an produzierten Tupeln des In-Network-Operators

$\bar{op}^{\mathcal{N}}$ im Zeitintervall Δt entspricht dann $out(\bar{op}^{\mathcal{N}}) = \sum_N out(\bar{op}_N^{\mathcal{N}})$ und die Anzahl an konsumierten Tupeln ergibt sich zu $in(\bar{op}^{\mathcal{N}}) = \sum_N in(\bar{op}_N^{\mathcal{N}})$. Für die Selektivitätsermittlung des Gesamtoperators muss somit die Gesamtheit der von dem Operator konsumierten und produzierten Datensätze herangezogen werden, d.h. die Selektivität von $\bar{op}^{\mathcal{N}}$ ist:

$$\sigma(\bar{op}^{\mathcal{N}}) = \frac{\sum_N out(\bar{op}_N^{\mathcal{N}})}{\sum_N in(\bar{op}_N^{\mathcal{N}})}$$

Diese vereinfachende Annahme stellt keine Einschränkung dar. Zwar kann sich der Energieverbrauch für eine einzelne Operation auf verschiedenen Knoten unterschiedlich entwickeln, für den Gesamtenergieverbrauch ist es allerdings unerheblich, auf welchem Knoten eine Operation durchgeführt wird. Generell ist eine nicht balancierte Auslastung der Knoten in einem WSN möglich (einzelne Knoten werden stärker belastet als andere), was wiederum beim Ausfall dieser zu einer reduzierten Gesamtlebensdauer des WSN führen kann. Dies zu vermeiden ist aber Aufgabe der verwendeten Operator-Algorithmen und wird bei der Kostenermittlung nicht berücksichtigt.

Komplexe Operatoren Im Allgemeinen leitet ein Operator alle von ihm erzeugten Datensätze an die bei ihm registrierten Operatoren weiter. Im Fall eines einfachen Operators bedeutet dies die Weiterleitung an exakt einen Operator. Mit der Überführung komplexer Operatoren in Mengen physischer Operatoren existieren allerdings auch Operatoren, welche über mehr als einen Ausgang verfügen (zum Beispiel für die Kommunikation zum Weiterleiten von Ergebnissen und zum Austauschen von Zwischenergebnissen mit Nachbarknoten). Die Menge der produzierten Daten in Bezug auf diese Ausgänge kann verschieden sein und muss im Kostenmodell entsprechend berücksichtigt werden.

Ein komplexer Operator $\bar{op}_i$ konsumiert $in(\bar{op}_i)$ Datensätze im Zeitintervall Δt und leitet Ergebnisse an den registrierten Operator $\bar{op}_j$ weiter. Somit ergibt sich eine Output-Rate von $out(\bar{op}_i \rightarrow \bar{op}_j)$ für den Operator $\bar{op}_i$ in Bezug auf den Operator $\bar{op}_j$. Die Selektivität für den komplexen Operator $\bar{op}_i$ in Bezug auf Operator $\bar{op}_j$ ist entsprechend

$$\sigma(\bar{op}_i \rightarrow \bar{op}_j) = \frac{out(\bar{op}_i \rightarrow \bar{op}_j)}{in(\bar{op}_i)}$$

Aktivierungshäufigkeit Sind sowohl die Produktivitätsraten aller Quellen als auch die Selektivitäten der einzelnen Operatoren bekannt, kann die Aktivierungshäufigkeit ϕ eines Operators $\bar{op}$ bezüglich einer Anfrage $\bar{q}$ im Zeitintervall Δt wie folgt bestimmt werden:

$$\phi(\bar{q}, \bar{op})_{\Delta t} = \begin{cases} out(\bar{op}) & pred_{\bar{q}}(\bar{op}) = \emptyset \\ \sum_{i \in pred_{\bar{q}}(\bar{op})} \sigma(i) \cdot \phi(\bar{q}, i)_{\Delta t} & pred_{\bar{q}}(\bar{op}) \neq \emptyset \end{cases} \tag{4.1}$$

$pred_{\bar{q}}(\bar{op})$ bezeichnet hierbei die Menge der direkten Vorgängerknoten des Operators $\bar{op}$ im Anfrageplan $\bar{q}$. Die Berechnung der Aktivierungshäufigkeit hängt lediglich von dem

Output der Quelloperatoren und der Selektivität der dazwischenliegenden Operatoren ab. Mit diesem Ansatz ist es möglich, alternative Anfragepläne auf Basis derselben statistischen Daten zu entwickeln und zu bewerten.

Ein Problem bei diesem Ansatz ist, dass die Selektivtät eines Operators möglicherweise durch andere Operatoren beeinflusst wird (die Selektivitäten der Operatoren sind voneinander abhängig). Ein Beispiel soll dies verdeutlichen: Es wird ein Ausführungsplan angenommen, in welchem zwei Filter-Operatoren hintereinander ausgeführt werden. Durch die Selektion des ersten Filters werden dabei sowohl die Input-Rate als auch die Datencharakteristika für den zweiten Operator geändert. Unabhängig davon, ob die beiden Filter über unterschiedlichen Attributen filtern, kann es passieren, dass durch den ersten Filter Tupel entfernt werden, welche den zweiten passieren bzw. durch diesen eventuell herausgefiltert würden. D.h., die Ausführung des ersten Operators hat Einfluss auf die Selektivität des nachfolgenden Operators und damit auch aller Nachfolgeoperatoren. Das Problem ist bereits aus dem Bereich der DBMS bekannt. Da bisher keine adäquate Lösung hierfür bekannt ist, wird zumeist die Unabhängigkeit der einzelnen Operatoren angenommen, was im Weiteren für diese Arbeit ebenfalls gelten soll.

4.2.1.2 Ausführungskosten

Neben der Ausführungshäufigkeit eines Operators sind die durch seine Ausführung entstehenden Kosten entscheidend. Im Fall der In-Network-Verarbeitung ist die für die Ausführung eines Operators $\bar{op}$ notwendige Energie $\mathsf{E}(\bar{op})$ (in μJ) von Interesse. Je nach verwendetem Operator lässt sich die für die Berechnungen benötigte Energie durch das einmalige Vermessen bzw. durch entsprechende approximative Verfahren ermitteln. Im Folgenden wird die Kostenermittlung/-berechnung für Quellen, innere Knoten und Senken betrachtet.

Quellen Quell-Operatoren erzeugen ausschließlich Daten. Die INQP-Komponente von *AnduIN* stellt zwei Arten von Quell-Operatoren bereit, *Sampling-Quellen* und *Netzwerk-Quellen*. *Sampling-Quellen* dienen der Erfassung von Daten auf Basis physischer Sensoren (zum Beispiel Sensoren für Temperatur oder Luftfeuchtigkeit). Es kann angenommen werden, dass die Kosten hierfür konstant sind und somit eine einmalige Vermessung der Operatoren ausreicht.

Netzwerk-Quellen überwachen eine Nachrichtenwarteschlange und werden nur beim Eintreffen neuer Nachrichten (für eine Anfrage) aktiviert. Erhält ein Sensorknoten eine Nachricht für einen Quell-Operator, dann wird diese Nachricht aus der Warteschlange entnommen und durch den Operator ein entsprechendes Tupel erzeugt, welches im Anschluss an die bei der Quelle registrierten Operatoren weitergeleitet wird. Im Weiteren sollen die Kostenapproximation für einen entsprechenden Operator beschrieben werden. Zur Vereinfachung werden für das Empfangen einer Nachricht im Folgenden die gleichen Kosten wie für das Versenden angenommen.

Normalerweise hängen die Energiekosten für das Senden und Empfangen einer Nach-

richt von der Länge der Nachricht ab. Diese setzt sich aus einem konstanten Overhead (für den Nachrichten-Header) und der Länge der eigentlichen Nachricht zusammen. Die Kosten, welche für das Senden eines Bytes anfallen, können durch einmaliges Messen bestimmt werden. Hierbei handelt es sich allerdings nur um einen Mittelwert, da Sensorknoten ihre Sendeleistung in Abhängigkeit von der Distanz zwischen den Knoten anpassen können.

Da sowohl die Größe des Nachrichtenheaders, als auch das Schema der untereinander zu versendenden Daten zum Zeitpunkt der Anfrageoptimierung bekannt sind, kann die benötigte Energiemenge entsprechend approximiert werden.

Im Fall von *AnduIN* gestaltet sich die Approximation recht einfach. Das auf den Sensorknoten eingesetzte Betriebssystem Feuerware versendet immer Datenpakete zu 62 Byte (6 Byte Header plus 56 Byte Daten), unabhängig davon, ob diese vollständig mit Daten gefüllt sind. Gemäß des verwendeten Protokolls werden Daten dabei bis zu 146 mal mit maximaler Leistung in Form von kurzen Bursts gesendet bis der Empfängerknoten die Daten empfangen hat[1]. In [107] wurde gezeigt, dass bei diesem Ansatz für das erfolgreiche Versenden eines Datenpaketes im Schnitt 73 Versuche notwendig sind. Sind die Kosten für das Initialisieren und das Versenden innerhalb eines Bursts bekannt (siehe Abschnitt 6.2.1 Tabelle 6.1), dann können die mittleren Kosten für das Versenden von maximal 56Byte Daten entsprechend abgeschätzt werden. Es bleibt daher nur noch zu ermitteln, wie viele Nachrichten für das Versenden der Teilergebnisse einer Anfrage notwendig sind.

Üblicherweise befinden sich Knoten, welche miteinander kommunizieren müssen, nicht in direkter Nachbarschaft zueinander. Nachrichten müssen über eventuell dazwischenliegende Knoten bis zum Zielknoten weitergeleitet werden. Die für den Transfer einer Nachricht notwendige Anzahl an Sprüngen h wird Hop-Distanz genannt. Im obigen Kostenmodell kommunizieren komplexe Operatoren eventuell über die entsprechenden Schnittstellen mit auf anderen Sensorknoten laufenden Instanzen derselben Operatoren. Die durchschnittliche Sprungweite für den Datentransfer zwischen zwei Instanzen eines komplexen Operators kann sich von der anderer komplexer Operatoren unterscheiden. Um dies im Kostenmodell zu berücksichtigen, wird für jeden einzelnen komplexen logischen Operator *op*, welcher über eine entsprechende Zerlegung mit Kommunikationsoperatoren verfügt, eine mittlere Hop-Distanz h_{op} eingeführt. Die Transferkosten für das Versenden einer Nachricht $\mathsf{E}_{transfer}(\bar{op}^{\mathcal{N}\downarrow})$ ($\bar{op}^{\mathcal{N}\downarrow}$ entspricht der Netzwerk-Quelle der physischen Zerlegung von *op*) können wie folgt abgeschätzt werden

$$\mathsf{E}_{transfer}(\bar{op}^{\mathcal{N}\downarrow}) = 2 \cdot \mathsf{E}_{publish} \cdot (h_{op} - 1)$$

Die Transferkosten werden im Rahmen der Kosten für das Ausführen einer Netzwerk-Quelle berücksichtigt. Jedesmal, wenn ein entsprechender Operator aktiviert wird, müssen die Daten zunächst von einem anderen Sensorknoten zu dieser Quelle transferiert worden sein. Somit ergeben sich für eine Netzwerk-Quelle $\bar{op}^{\mathcal{N}\downarrow}$ die folgenden

[1] das burst-artige Versenden dient dabei der Reduktion von Kollisionen aufgrund des geteilten Transportmediums

Kosten

$$\mathsf{E}(\bar{op}^{\mathcal{N}\downarrow}) = \mathsf{E}_{publish} + \mathsf{E}_{transfer}(\bar{op}^{\mathcal{N}\downarrow})$$

Der Fanout als Maß für die Verteilung von Daten eines Sensorknotens im WSN wird somit implizit über die Output-Rate der entsprechenden Netzwerk-Quellen beschrieben. Verteilt ein Operator zum Beispiel seine Daten an drei weitere Sensorknoten, so erfolgen entsprechend drei Aktivierungen der Netzwerk-Quellen auf den Empfängerknoten.

Die Kosten für den Knoten der Basisstation als Teil der DSMS-Komponente müssen gesondert betrachtet werden. Da dieser üblicherweise mit einer externen Energiequelle versehen ist, kann die für das Empfangen von Nachrichten notwendige Energie vernachlässigt werden. Dennoch müssen die Daten von den einzelnen Sensorknoten zum Knoten der Basisstation transportiert werden, so dass für die Netzwerk-Quell-Operatoren $\bar{op}^{\mathcal{L}\downarrow}$ der Basisstation die folgenden Kosten anfallen:

$$\mathsf{E}(\bar{op}^{\mathcal{L}\downarrow}) = \mathsf{E}_{transfer}(\bar{op}^{\mathcal{L}\downarrow}) = 2 \cdot \mathsf{E}_{publish} \cdot (h-1)$$

Die mittlere Sprungweite h entspricht hierbei der mittleren Distanz zur Basisstation. h kann wie folgt abgeschätzt werden: Der zentrale Knoten versendet eine Broadcast-Nachricht. Beim Weiterleiten dieser Nachricht werden in den passierten Knoten entsprechende Zähler inkrementiert. Haben alle Knoten die Nachricht erhalten, dann wird die Nachricht von den Blattknoten zur Basisstation zurückgeschickt. Während des Zurücksendens werden die auf den Knoten befindlichen Zähler aggregiert und die Nachrichten weitergeleitet. Der zentrale Knoten kann anschließend aus der so ermittelten Anzahl der Knoten im WSN und den notwendigen Hops h ermitteln. Die mittlere Hop-Distanz für komplexe Operatoren mit Datenaustausch zwischen den Sensorknoten kann über entsprechende Erfahrungswerte ermittelt werden (bei bekannter Netzwerkstruktur, -topologie und WSN-Größe).

Üblicherweise befinden sich die Sensorknoten eines WSN im Sleep-Mode. Die Aktivierung einer Quelle entspricht damit der Reaktivierung der Sensorknoten. Die Kosten für einen Wake-Up sind nicht zu vernachlässigen, können aber durch einmaliges Messen bestimmt werden und bei den Kosten der Quell-Operatoren berücksichtigt werden (da diese immer am Anfang einer Verarbeitungssequenz liegen). Werden mehrere Quellen auf einem Sensorknoten während einer Wach-Phase gelesen, so müssen die für das Aufwachen notwendigen Kosten nur einmal betrachtet werden.

Senken Die Kosten für die Senken können ähnlich wie bei den Netzwerkquellen über die Menge der zu versendenden Daten und die damit verbundenen Kosten ermittelt werden. Hierbei sind allerdings lediglich die Kosten für das Versenden zu berechnen. Die Kosten für den Transfer und das Empfangen wurden bereits in die Berechnung der Kosten für die entsprechenden Netzwerk-Quellen integriert.

Innere Knoten Für die Bestimmung der Kosten eines inneren Operators muss der Berechnungsaufwand der implementierten Verfahren herangezogen werden. Für Algorithmen mit konstantem Berechnungsaufwand (Filter, Projektion) ist eine einmalige

Erfassung der Kosten für einen neu eintreffenden Datensatz ausreichend. Die für einen Operator mit nicht konstantem Berechnungsaufwand anfallenden Kosten müssen entsprechend approximiert werden bzw. es muss auf entsprechende Statistiken zurückgegriffen werden. Die Kostenermittlung für einen einfachen Operator mit nicht konstanten Kosten soll am Beispiel des *Minimum-Operators* $\bar{op}_{min}$ gezeigt werden.

Beispiel 4.1 Der Minimum-Operator liefert beim Eintreffen eines neuen Tupels den minimalen Wert über die letzten k Tupel des Datenstromes. w bezeichnet dabei die maximale Anzahl von Tupeln im betrachteten Zeitfenster. Ein neu eintreffender Wert wird mit dem aktuellen Minimum verglichen. Falls dieser kleiner ist, wird er als neues Minimum verwendet. Anschließend wird der Wert in das Fenster integriert. Durch das Einfügen des neuen Wertes wird der älteste Wert aus dem Fenster verdrängt. Entspricht der verdrängte Wert nicht dem aktuellen Minimum, dann ist die Verarbeitung für diese Aktivierung an dieser Stelle beendet. Handelt es sich bei dem aus dem Fenster verdrängten Wert jedoch um das aktuelle Minimum, dann muss das Fenster auf den neuen minimalen Wert hin untersucht werden. Im schlimmsten Fall ist hierzu das vollständige Fenster zu testen.

Da zum Zeitpunkt der Kostenbewertung der Anfrage die Fenstergröße w bekannt ist, können die voraussichtlich entstehenden Energiekosten unter Verwendung von zwei Energiemessungen E_1 und E_2 mit Fenstergrößen w_1 bzw. w_2 durch nachstehende lineare Gleichung abgeschätzt werden:

$$\mathsf{E}(\bar{op}_{min}, w) = \left(\frac{\mathsf{E}_2 - \mathsf{E}_1}{w_2 - w_1}\right) \cdot w - \left(\frac{\mathsf{E}_2 - \mathsf{E}_1}{w_2 - w_1}\right) \cdot w_1 + \mathsf{E}_1$$

In Abschnitt 6.2.1.2 wird gezeigt, dass diese Art der Abschätzung gute Ergebnisse liefert insofern die Datenverteilung bekannt ist. ■

Der Energieverbrauch von komplexen Algorithmen, für die sich keine einfachen approximativen Verfahren finden lassen, kann zum Beispiel über den Vergleich mit entsprechenden Statistiken ermittelt werden.

4.2.1.3 Anfragekosten

Sind die Ausführungshäufigkeit eines Operators und die anfallenden Ausführungskosten bekannt, dann können die Kosten für die Ausführung des physischen Anfrageplanes $\bar{q}$ im WSN im Zeitintervall Δt wie folgt bestimmt werden:

$$\mathsf{E}(\bar{q}, \Delta t) = \sum_{\bar{op} \in \bar{q}} \mathsf{E}(\bar{op}) \cdot \phi(\bar{q}, \bar{op})_{\Delta t} \tag{4.2}$$

Im Folgenden sollen zwei Beispiele die Kostenapproximation auf Basis des hier vorgestellten Kostenmodells verdeutlichen. Tabelle 4.1 gibt einen Überblick über die in den Beispielen verwendeten Operatoren.

Beispiel 4.2 Jeder Sensorknoten im WSN erfasst fünf Mal pro Minute die Attribute A und B (Samplingoperator $\bar{\zeta}_{A,B}$). Anschließend werden die so erzeugten Tupel gefiltert. Lediglich Tupel, für die $B > x$ gilt, passieren $\bar{\sigma}^{\mathcal{N}}_{B>x}$. Bei diesem Beispiel bearbeiten

Symbol	Operator
$\zeta_{A,B}$	Sampling-Operator (erfasst Attribute A und B)
$\bar{\sigma}^{\mathcal{N}}_{B>x}$	Filter Operator (Prädikat $B > x$)
$\overline{conn}^{\mathcal{N}\uparrow}_{A,B}$	INQP-DSMS-Schnittstelle - Senke (sendet Attribute A und B)
$\overline{conn}^{\mathcal{L}\downarrow}_{A,B}$	INQP-DSMS-Schnittstelle - Quelle (empfängt Attribute A und B)
$a\bar{g}g^{\mathcal{N}}_{B}$	Aggregatoperator über Attribut B
$a\bar{g}g^{\mathcal{N}\downarrow}_{A,B}$	In-Network-Quelle für $a\bar{g}g^{\mathcal{N}}_{B}$
$a\bar{g}g^{\mathcal{N}\uparrow}_{A,B}$	In-Network-Senke für $a\bar{g}g^{\mathcal{N}}_{B}$
$\overline{conn}^{\mathcal{N}\uparrow}_{a\bar{g}g_{A,B}}$	Senke für $a\bar{g}g^{\mathcal{N}}_{B}$ zum Senden an die DSMS-Komponente
$\overline{conn}^{\mathcal{L}\downarrow}_{a\bar{g}g_{A,B}}$	Quelle für $a\bar{g}g^{\mathcal{N}}_{B}$ zum Empfangen auf der DSMS-Komponente

Tabelle 4.1: In Beispielen verwendete Operatoren

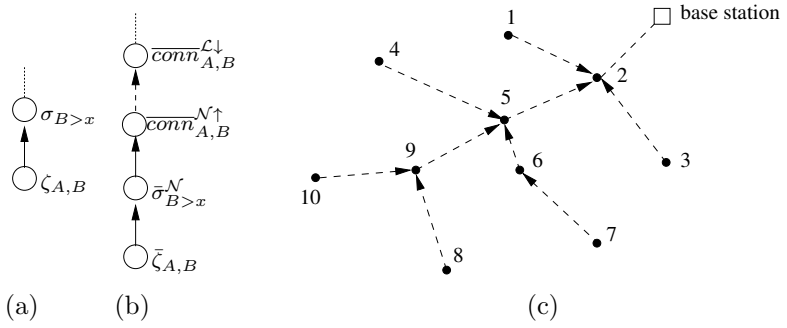


Abbildung 4.5: Beispiel 4.2: (a) logischer Anfrageplan, (b) physischer Ausführungsplan, (c) Netzwerkausprägung (Topologie)

die Sensorknoten ihre Anfrageteile unabhängig von den Ergebnissen benachbarter Knoten. Zur Erinnerung: Alle Sensorknoten führen exakt den gleichen Anfrageplan aus. Abschließend senden sie ihre Ergebnisse an die zentrale Instanz, welche die weitere Verarbeitung vornimmt.

In Abbildung 4.5 ist der In-Network-Teil des zu verarbeitenden logischen Anfrageplanes und eine mögliche Transformation in einen physischen Ausführungsplan dargestellt. Zusätzlich zeigt Abbildung 4.5 eine angenommene Netzwerkausprägung mit zehn Sensorknoten und einer Basisstation. Im Netzwerk werden somit pro Minute 50 Werte gesampelt (zehn Knoten zu fünf Samples pro Minute). Auf Basis der Abbildung kann für die Kommunikation mit der zentralen Instanz die mittlere Hop-Distanz bestimmt werden $2.8 = (1 + 2 + 2 + 2 + 3 + 3 + 3 + 4 + 4 + 4)/10$. Für den Filter-Operator wird eine Selektivität von 0.5 angenommen, d.h. im Mittel passiert lediglich jedes zweite Tupel den Operator. Somit kann für eine Minute Betrieb des WSN die folgende

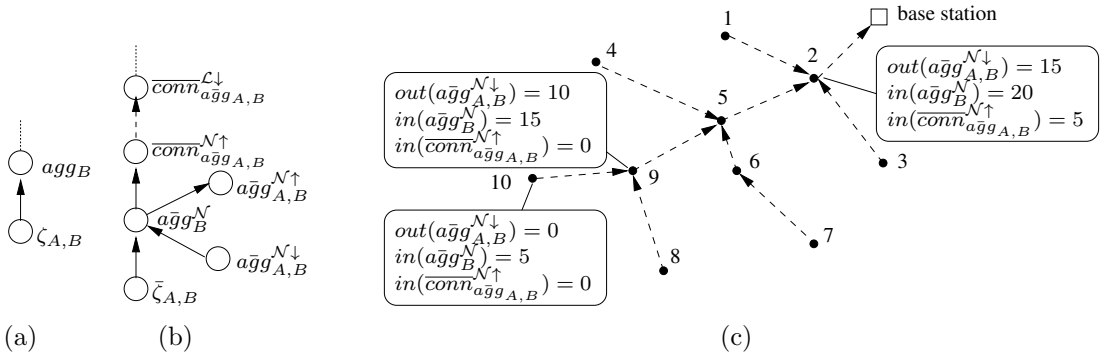


Abbildung 4.6: Beispiel 4.3: (a) logischer Anfrageplan, (b) physischer Ausführungsplan, (c) Netzwerkausprägung (Topologie)

Energie-Abschätzung vorgenommen werden:

$$\begin{aligned}
\mathsf{E}(\bar{q}_1) &= 50 \cdot \mathsf{E}(\bar{\zeta}_{A,B}) + 50 \cdot \mathsf{E}(\bar{\sigma}^{\mathcal{N}}_{B>x}) + 50 \cdot 0.5 \cdot \mathsf{E}(\overline{conn}^{\mathcal{N}\uparrow}_{A,B}) + 50 \cdot 0.5 \cdot \mathsf{E}(\overline{conn}^{\mathcal{L}\downarrow}_{A,B}) \\
&= 50 \cdot (\mathsf{E}(\bar{\zeta}_{A,B}) + \mathsf{E}(\bar{\sigma}^{\mathcal{N}}_{B>x})) + 25 \cdot (\mathsf{E}_{publish} + \mathsf{E}_{transfer}(\overline{conn}^{\mathcal{L}\downarrow}_{A,B})) \\
&= 50 \cdot (\mathsf{E}(\bar{\zeta}_{A,B}) + \mathsf{E}(\bar{\sigma}^{\mathcal{N}}_{B>x})) + 25 \cdot (\mathsf{E}_{publish} + 2 \cdot \mathsf{E}_{publish} \cdot (h_{\overline{conn}^{\mathcal{L}\downarrow}_{A,B}} - 1)) \\
&= 50 \cdot (\mathsf{E}(\bar{\zeta}_{A,B}) + \mathsf{E}(\bar{\sigma}^{\mathcal{N}}_{B>x})) + 115 \cdot \mathsf{E}_{publish}
\end{aligned}$$

Die exakten Kosten für das Sampling und das Filtern können durch Messungen bestimmt und entsprechend eingesetzt werden. Für das Senden zur zentralen Instanz fallen im gesamten Netzwerk 115 Nachrichten an. Ein einfaches Auszählen zeigt, dass die Anzahl an zu sendenden Nachrichten korrekt ist. Da das Nachrichten-Schema bekannt ist (die Attribute A und B müssen versendet werden), lassen sich Kosten für den Versand einer Nachricht $\mathsf{E}_{publish}$ ebenfalls bestimmen. ■

Im Abschnitt 6.2.1 werden Evaluationsergebnisse für ein ähnliches Beispiel mit realen Messwerten präsentiert.

Beispiel 4.3 Im Folgenden soll ein komplexer In-Network-Operator als Teil der Anfrageverarbeitung betrachtet werden. Die Anfrage besteht dabei aus dem Sampling der Attribute A und B und einer anschließenden In-Network-Aggregation, wie sie zum Beispiel mit TAG [147] präsentiert wurde. Im Gegensatz zum einfachen Aggregationsoperator, bei welchem lediglich einfache Werte über einem Datenstrom aggregiert werden, ist das Ziel der In-Network-Aggregation das Berechnen eines Aggregates über den Daten aller Knoten des Sensornetzwerkes. Ein typisches Beispiel ist das Finden des Maximums über allen Knoten im WSN. Hier wird pro Zeiteinheit lediglich ein einziger Wert zurückgeliefert. Bei dieser Form der Aggregation warten die inneren Knoten des WSN auf die Teilergebnisse ihrer Kindknoten und aggregieren deren Ergebnisse beim Eintreffen mit ihren eigenen Zwischenergebnissen. Das Ergebnis wird anschließend weiter in Richtung zentrale Instanz geleitet.

Abbildung 4.6 zeigt einen logischen Anfrageplan und eine mögliche physische Aus-

prägung. Für die Kommunikation mit den Nachbarknoten verfügt der In-Network Aggregat-Operator $a\bar{g}g_B^{\mathcal{N}}$ (es wird über dem Attribut B aggregiert) über eine zusätzliche Netzwerk-Quelle $a\bar{g}g_{A,B}^{\mathcal{N}\downarrow}$ und eine zusätzliche Senke $a\bar{g}g_{A,B}^{\mathcal{N}\uparrow}$.

Die Topologie des WSN entspricht der aus dem vorhergehenden Beispiel, d.h. es handelt sich um ein WSN mit 10 Knoten. Jeder Knoten erfasst fünf Mal pro Minute die Attribute A und B, $out(\bar{\zeta}_{A,B}) = 50$. Die Output-Rate von Operator $a\bar{g}g_B^{\mathcal{N}}$ mit Bezug auf Operator $a\bar{g}g_{A,B}^{\mathcal{N}\uparrow}$ ist 45 (alle Knoten, außer Knoten 2, leiten 5 Tupel an ihren Elternknoten weiter). Entsprechend produziert der Operator $a\bar{g}g_{A,B}^{\mathcal{N}\downarrow}$ 45 Tupel (innere Knoten empfangen die Nachrichten ihrer Nachbarn). Die Output-Rate von $a\bar{g}g_B^{\mathcal{N}}$ mit Bezug auf $\overline{conn}_{a\bar{g}g_{A,B}}^{\mathcal{N}\uparrow}$ ist 5, da ausschließlich Knoten 2 Ergebnisse an den Operator $\overline{conn}_{a\bar{g}g_{A,B}}^{\mathcal{N}\uparrow}$ der zentralen Instanz weiterleitet (fünf Mal pro Minute). Somit ergeben sich die folgenden Selektivitäten: $\sigma(a\bar{g}g_B^{\mathcal{N}} \rightarrow \overline{conn}_{a\bar{g}g_{A,B}}^{\mathcal{N}\uparrow}) = \frac{5}{50+45} = \frac{1}{19}$ und $\sigma(a\bar{g}g_B^{\mathcal{N}} \rightarrow a\bar{g}g_{A,B}^{\mathcal{N}\uparrow}) = \frac{45}{50+45} = \frac{9}{19}$.
Der Gesamtenergieverbrauch des In-Network-Teils der Beispielanfrage ergibt sich nach obigem Kostenmodell somit wie folgt:

$$\begin{aligned}\mathsf{E}(\bar{q}_2) &= 50 \cdot \mathsf{E}(\bar{\zeta}_{A,B}) + 95 \cdot \mathsf{E}(a\bar{g}g_B^{\mathcal{N}}) + 95 \cdot \frac{9}{19} \cdot \mathsf{E}(a\bar{g}g_{A,B}^{\mathcal{N}\uparrow}) + 45 \cdot \mathsf{E}(a\bar{g}g_{A,B}^{\mathcal{N}\downarrow}) \\ &\quad +95 \cdot \frac{1}{19} \cdot \mathsf{E}(\overline{conn}_{a\bar{g}g_{A,B}}^{\mathcal{N}\uparrow})\end{aligned}$$

Sowohl die Kommunikation zwischen den Operatoren $a\bar{g}g_{A,B}^{\mathcal{N}\uparrow}$ und $a\bar{g}g_{A,B}^{\mathcal{N}\downarrow}$ als auch die Kommunikation zwischen $\overline{conn}_{a\bar{g}g_{A,B}}^{\mathcal{N}\uparrow}$ und $\overline{conn}_{a\bar{g}g_{A,B}}^{\mathcal{L}\downarrow}$ ist Teil des komplexen Aggregationsoperators. Für diesen beträgt die durchschnittliche Hop-Distanz 1 (Sprungweite zwischen zwei verarbeitenden Knoten). Somit ergeben sich die folgenden Kosten:

$$\begin{aligned}\mathsf{E}(\bar{q}_2) &= 50 \cdot \mathsf{E}(\bar{\zeta}_{A,B}) + 95 \cdot \mathsf{E}(a\bar{g}g_B^{\mathcal{N}}) + 45 \cdot \mathsf{E}_{publish} + 45 \cdot (\mathsf{E}_{publish} + 2 \cdot \mathsf{E}_{publish} \cdot (1-1)) \\ &\quad +5 \cdot \mathsf{E}_{publish} + 5 \cdot 2 \cdot \mathsf{E}_{publish} \cdot (1-1) \\ &= 50 \cdot \mathsf{E}(\bar{\zeta}_{A,B}) + 95 \cdot \mathsf{E}(a\bar{g}g_B^{\mathcal{N}}) + 95 \cdot \mathsf{E}_{publish}\end{aligned}$$

Die Kosten für das Sampling, die Aggregation und das Senden können, wie oben beschrieben, abgeschätzt werden. ■

4.2.1.4 Bestimmung von Tupelraten

Generell lässt sich die Selektivität eines Operators über entsprechende Statistiken (Input-/Output-Rate) erfassen. Einfache Operatoren hängen dabei im Wesentlichen von der Verteilung der eingehenden Daten ab. Entsprechende Betrachtungen (zum Beispiel Histogramme zur Bestimmung von Häufigkeitsverteilungen [112,167,184]) wurden bereits im Kontext der Anfrageoptimierung von DBMS untersucht und sollen hier nicht weiter ausgeführt werden.

Die Input- bzw. Output-Raten der im Kostenmodell eingeführten, zusätzlichen Kommunikationsoperatoren als Teil der komplexen Operatoren hängt zusätzlich noch von der verwendeten Netzwerktopologie und dem Fanout der einzelnen Sensorknoten ab.

Prinzipiell ist es auch hier möglich, entsprechende Statistiken zu erfassen, um geeignete Approximationen für diese berechnen zu können. In *AnduIN* werden durch Statistik-Monitore die Input- bzw. Output-Raten jedes einzelnen Operators erfasst. Da die Anfrageoptimierung auf der zentralen Instanz stattfindet, ist auch die Verwaltung großer statistischer Datenmengen kein Problem.

Ein anderer Weg ist eine analytische Approximation der Operator-Raten. Im Folgenden soll diese Möglichkeit am Beispiel der In-Network-Aggregation aus Beispiel 4.3 demonstriert werden. Als Netzwerktopologie wird ein balancierter Baum der Höhe H angenommen. Der Fanout pro Kindknoten im Beispiel der In-Network-Aggregation ist 1. Jeder Knoten hat f Kindknoten. Ein Elternknoten erhält somit Daten von f Kindknoten. Mit diesen Annahmen können die Output-Rate $out(a\bar{g}g^{\mathcal{N}\downarrow})$ und die Selektivität $\sigma(a\bar{g}g^{\mathcal{N}})$ des Aggregationsoperators wie folgt approximiert werden:

$$
\begin{aligned}
out(a\bar{g}g^{\mathcal{N}\downarrow}) &= \left(\sum_{i=0}^{h-1} f^i - 1\right) \cdot \frac{out(\bar{op})}{\sum_{i=0}^{h-1} f^i} = out(\bar{op}) - \frac{out(\bar{op})}{\sum_{i=0}^{h-1} f^i} \\
\sigma(a\bar{g}g^{\mathcal{N}} \rightarrow \overline{conn}_{a\bar{g}g}^{\mathcal{N}\uparrow}) &= \frac{out(\bar{op})/(\sum_{i=0}^{h-1} f^i)}{(out(\bar{op}) + out(a\bar{g}g^{\mathcal{N}\downarrow}))} \\
\sigma(a\bar{g}g^{\mathcal{N}} \rightarrow a\bar{g}g^{\mathcal{N}\uparrow}) &= \frac{out(a\bar{g}g^{\mathcal{N}\downarrow})}{(out(\bar{op}) + out(a\bar{g}g^{\mathcal{N}\downarrow}))}
\end{aligned}
$$

$\bar{op}$ bezeichnet den direkten Vorgänger-Operator von $a\bar{g}g^{\mathcal{N}}$, welcher nicht der Netzwerk-Quelle $a\bar{g}g^{\mathcal{N}\downarrow}$ entspricht. In Beispiel 4.3 ist dies $\bar{op} = \bar{\zeta}_{A,B}$. Die Approximation der Outrate bzw. der Selektivitäten $outlier(a\bar{g}g^{\mathcal{N}\downarrow})$, $\sigma(a\bar{g}g^{\mathcal{N}} \rightarrow \overline{conn}_{a\bar{g}g}^{\mathcal{N}\uparrow})$ und $\sigma(a\bar{g}g^{\mathcal{N}} \rightarrow a\bar{g}g^{\mathcal{N}\uparrow})$ hängt hierbei ausschließlich von der Output-Rate des Operators $\bar{op}$ und der Netzwerktopologie ab.

Der Fanout wird beim vorgestellten Kostenmodell über die Output-Raten der entsprechenden Kommunikationsoperatoren abgebildet. Häufig finden sich in der Literatur auch weitere Parameter zur Berechnung von Kommunikationskosten. So gibt zum Beispiel die mittlere Verlust-Rate von Datenpaketen an, welche Menge an Daten im Mittel aufgrund von Kommunikationsproblemen (zum Beispiel Paketkollisionen) im WSN verloren geht. Da verlorene Pakete nicht zu einer Aktivierung von Netzwerk-Quellen führen können, wird bei ausschließlicher Verwendung von Statistiken die Verlust-Rate ebenfalls über die Output-Rate beschrieben.

4.2.2 Ausführungskosten der zentralen Komponente

Da die zentrale Komponente im Allgemeinen über eine externe Energieversorgung verfügt, sind auf dieser Energiekosten nicht von Interesse. Die Energiekosten für die Verarbeitung auf der zentralen Komponente werden entsprechend 0 gesetzt. Aufgrund der möglichen Mehrbelastung durch die Parallelverarbeitung einer Vielzahl von Anfragen stehen auf der zentralen Komponente andere Optimierungsparameter im Vordergrund

(zum Beispiel Datenqualität, Speicherauslastung, Externspeicherzugriffe). Im Folgenden wird an dieser Stelle stellvertretend die Maximierung des Datensatzdurchsatzes demonstriert.

Der Durchsatz hängt von zwei Parametern ab: (i) der Anzahl der Datensätze, die ein Operator pro Zeiteinheit als Eingabe erhält und (ii) die Anzahl der Datensätze, die ein Operator pro Zeiteinheit maximal verarbeiten kann. Sind die Selektivitäten der inneren Operatoren eines Anfrageplanes und die Output-Raten aller Senken bekannt, so kann die Aktivierungshäufigkeit für jeden Operator nach Formel 4.1 bestimmt werden.

Der maximale Datendurchsatz eines Operators kann auf Basis des Inversen der Verarbeitungskosten $\mathsf{C}(\bar{op})$, welche für die Verarbeitung eines einzelnen Tupels anfallen, ermittelt werden. Benötigt ein Operator $0.1ms$ für die Verarbeitung eines einzelnen Datensatzes, dann ergibt sich ein maximaler Durchsatz von $(0.001\frac{s}{T})^{-1} = 1000\frac{T}{s}$ (T bezeichnet die Tupelanzahl). Die für die Verarbeitung eines Tupels notwendige mittlere CPU-Zeit kann wiederum durch einmaliges Vermessen bestimmt werden. Für Operatoren mit einem konstanten Berechnungsaufwand ist ein einmaliges Vermessen ausreichend. Für Operatoren mit nicht konstantem Aufwand müssen geeignete approximative Verfahren entwickelt werden, mit denen der Durchsatz abgeschätzt werden kann (ähnlich der Berechnung des Energiebedarfs für die Verarbeitung eines Tupels im WSN). Es muss beachtet werden, dass die Verarbeitungsgeschwindigkeit von Datensätzen abhängig vom verwendeten System ist. Alle Messungen müssen daher auf demselben System durchgeführt werden. Da für eine Optimierung nur der relative Unterschied zwischen den einzelnen Operatoren von Interesse ist, können die so erfassten Werte letztendlich auf beliebigen Systemen zur Optimierung eingesetzt werden.

Da die Anzahl von Tupeln, welche durch die Quellen und Senken des auf der zentralen Komponente ablaufenden Anfrageteils pro Zeiteinheit zur Verfügung gestellt werden, durch den Output des In-Network-Teils bzw. durch die Anfrage selbst definiert sind und nicht durch eine Optimierung des zentral verarbeiteten Anfrageteils beeinflusst werden können, sind lediglich die für die inneren Knoten anfallenden Kosten für die Optimierung von Interesse. Daher können die Verarbeitungskosten für Quellen und Senken auf der zentralen Komponente vernachlässigt werden.

Die Gesamtkosten für die Ausführung einer Anfrage $\bar{q}$ auf der zentralen Komponente können somit folgendermaßen abgeschätzt werden:

$$\mathsf{C}(\bar{q}, \Delta t) = \sum_{\bar{op} \in \bar{q}^{\mathcal{L}} \setminus \bar{\mathbf{S}} \setminus \bar{\mathbf{z}}} \mathsf{C}(\bar{op}) \cdot \phi(\bar{q}, \bar{op})_{\Delta t} \tag{4.3}$$

Die so ermittelten Verarbeitungkosten $\mathsf{C}(\bar{q}, \Delta t)$ für den Anfrageplan sind dimensionslos (ohne Einheit, ähnlich wie zum Beispiel der Wirkungsgrad in der Physik).

Wie bereits erwähnt, existieren zwischen den einzelnen Operatoren in *AnduIN* keine Tupelpuffer. Auf der zentralen Komponente feuert jede Tupelquelle fortwährend, solang keiner der empfangenden Operatoren aufgrund zu hoher Auslastung blockiert. Ein Operator erfährt genau dann eine zu hohe Auslastung, wenn $\mathsf{C}(\bar{op}) \cdot \phi(\bar{q}, \bar{op})_{\Delta t} > 1$ gilt, d.h. wenn mehr Tupel zur Verarbeitung anliegen, als der Operator verarbeiten kann.

Prinzipiell ist es möglich, dass auch Pläne, welche weniger Ausführungskosten verursachen als andere, zu einer Blockierung der Verarbeitung im System führen. Ein einfaches Beispiel soll dies verdeutlichen:

Beispiel 4.4 Gegeben sei ein logischer Anfrageplan mit zwei inneren Operatoren op_1 und op_2. Am Quell-Operator gehen 10 Tupel pro Sekunde ein. op_1 wird in den physischen Operator $\bar{op}_1$ und op_2 in $\bar{op}_2$ übersetzt. Weiterhin sind die folgenden Ausführungskosten und Selektivitäten gegeben: $\mathsf{C}(\bar{op}_1) = 0.4$, $\sigma(\bar{op}_1) = 0.1$ und $\mathsf{C}(\bar{op}_2) = 0.1$, $\sigma(\bar{op}_2) = 0.3$.

Damit lassen sich die beiden Ausführungspläne $\bar{q}_{\bar{op}_1 \to \bar{op}_2}$ und $\bar{q}_{\bar{op}_2 \to \bar{op}_1}$ ($\bar{op}_1 \to \bar{op}_2$ entspricht der Ausführungsreihenfolge) mit nachstehenden Kosten erstellen:

$$\begin{aligned} \mathsf{C}(\bar{q}_{\bar{op}_1 \to \bar{op}_2}) &= 10\frac{T}{s} \cdot 0.4\frac{s}{T} + 10 \cdot 0.1\frac{T}{s} \cdot 0.1\frac{s}{T} = 4.1 \\ \mathsf{C}(\bar{q}_{\bar{op}_2 \to \bar{op}_1}) &= 10\frac{T}{s} \cdot 0.1\frac{s}{T} + 10 \cdot 0.3\frac{T}{s} \cdot 0.4\frac{s}{T} = 2.8 \end{aligned}$$

Plan $\bar{q}_{\bar{op}_2 \to \bar{op}_1}$ verursacht somit eine geringere Last als Plan $\bar{q}_{\bar{op}_1 \to \bar{op}_2}$. Allerdings kann die Ausführung von $\bar{op}_1$ in Plan $\bar{q}_{\bar{op}_2 \to \bar{op}_1}$ zum Blockieren ($10 \cdot 0.3\frac{T}{s} \cdot 0.6\frac{s}{T} > 1$) in *AnduIN* führen. ■

Wird ein Operator bei der anschließenden Ausführung tatsächlich vollständig ausgelastet, dann führt dies zu einer verringerten Output-Rate der Anfrage. Um dies zu verhindern, wird folgende Nebenbedingung für die Anfrageoptimierung definiert:

$$\forall \bar{op} \in \bar{q} : \mathsf{C}(\bar{op}) \cdot \phi(\bar{op})_{\Delta t} \leq 1$$

Es muss somit für alle physischen Operatoren einer Anfrage gelten, dass sie die weitere Verarbeitung der Anfrage nicht blockieren. Für das obige Beispiel bedeutet dies, dass Plan $\bar{q}_2$, obwohl er die geringeren Gesamtkosten verursacht, nicht optimal ist und somit auch nicht zur Ausführung gebracht wird.

Ein Problem besteht für den Fall, dass es nicht möglich ist, einen Plan zu entwickeln, welcher keine blockierenden Operatoren enthält. Einer der entwickelten Pläne muss aber zur Ausführung gebracht werden! Die vollständige Auslastung eines einzelnen Operators hat zur Folge, dass die Quellen das Produzieren entsprechend drosseln und weniger Tupel in die laufenden Anfragen gepusht werden. Im Fall von *AnduIN* werden eingehende Datensätze, welche aufgrund von voll ausgelasteten Operatoren nicht verarbeitet werden können, in den Quellen gepuffert (zum Beispiel als Teil der verwendeten Socket-Puffer, die der zentralen Komponente als Datenstromquellen dienen). Damit ergeben sich die folgenden Konsequenzen:

- Aufgrund der verringerten Output-Raten einzelner Quellen verringert sich der Durchsatz auf der zentralen Komponente.
- Da nur noch ein Teil der eintreffenden Datensätze erfolgreich bearbeitet werden kann, (die verbleibenden Datensätze werden entweder verzögert verarbeitet oder bei einem Pufferüberlauf verworfen), sinkt die Ergebnis-Qualität.

D.h. zusätzlich zum eigentlichen Datendurchsatz müsste im Fall von *AnduIN* beim Erreichen der maximalen Auslastung von Operatoren eine weitere Optimierungsdimension, die Ergebnis-Güte, betrachtet werden (ähnlich wie in [185, 186]). Ein einfaches Maß für letztere ist das Verhältnis aus der Anzahl der zu verarbeitenden und der Anzahl der tatsächlich verarbeiteten Datensätze. Da die Datenqualität bereits im Kontext von Techniken wie Load-Shedding und Sampling intensiv untersucht wurde, soll darauf im Weiteren nicht eingegangen werden.

4.2.3 Multikriterielle Anfrageoptimierung

Die gleichzeitige Optimierung mehrerer Zielfunktionen wird allgemein unter dem Begriff der multi-kriteriellen Optimierung zusammengefasst. Betrachtet man ein zweidimensionales Optimierungsziel wie im vorliegenden Fall, dann spricht man auch von bikriterieller Optimierung. Formal kann das im vorangegangenen Abschnitt beschriebene Optimierungsproblem wie folgt definiert werden:

$$vmin_{\bar{q}_i \in \bar{\mathbf{Q}}} MOP(\bar{q}_i) := vmin_{\bar{q}_i \in \bar{\mathbf{Q}}} [\mathsf{E}(\bar{q}_i), \mathsf{C}(\bar{q}_i)]$$

Im Allgemeinen versucht man bei der multikriteriellen Optimierung sämtliche Zielfunktionen zu minimieren (*vmin* meint dabei die Minimierung eines Vektors). Sollen Zielfunktionen maximiert werden, so kann dies durch Negation der zu optimierenden Funktion erreicht werden.

Für die Lösung multikriterieller Optimierungsprobleme gibt es zwei grundsätzliche Herangehensweisen:

- Das multikriterielle Problem wird auf ein einkriterielles Problem abgebildet, welches anschließend mit einem einkriteriellen Optimierungsverfahren gelöst werden kann.
- Als Ergebnis echter multikriterieller Optimierung wird eine Menge (pareto-)optimaler Lösungen bestimmt. Aus dieser Menge kann anschließend der beste Kompromiss zur Lösung des Problems identifiziert werden.

Im Folgenden sollen beide Ansätze sowie ihre Vor- und Nachteile kurz vorgestellt werden. Beide Ansätze wurden in Form eines speziellen Verfahrens als Teil der Planauswahl in *AnduIN* integriert.

4.2.3.1 Methode der gewichteten Summe

Ein einfacher Ansatz zur Lösung eines MOP ist dessen Transformation in ein einkriterielles Problem. Hierzu werden die Werte der verschiedenen Zielfunktionen mit vorab definierten Gewichten multipliziert und aufsummiert. Diese gewichtetete Summe

kann dann anschließend mit einem Optimierungsverfahren für eindimensionale Probleme gelöst werden. Angewandt auf das oben definierte bikriterielle Optiimierungsproblem bedeutet dies, dass die Summe $\lambda_1 \mathsf{E}(\bar{q}_i) + \lambda_2 \mathsf{C}(\bar{q}_i)$ optimiert werden muss. Es gilt somit:

$$MOP(\bar{q}_i) = min_{\bar{q}_i \in \bar{\mathbf{Q}}}[\lambda_1 \mathsf{E}(\bar{q}_i) + \lambda_2 \mathsf{C}(\bar{q}_i)]$$

wobei λ_1 und λ_2 die Gewichte der einzelnen Optimierungsziele bezeichnen und $\lambda_1 + \lambda_2 = 1$, mit $\lambda_i \geq 0$ gilt.

Einer der wesentlichen Nachteile dieser Methode ist, dass A-Priori-Wissen bezüglich der Gewichte, welches die Wichtigkeit der einzelnen Optimierungsziele zum Ausdruck bringt, benötigt wird. Dies heißt, der Kompromiss zwischen den verschiedenen Zielfunktionen muss bereits vor der eigentlichen Optimierungsphase bekannt sein. Eine Betrachtung anderer Lösungen ist nur durch eine erneute Optimierung mit anderen Gewichten möglich. Im Zusammenhang mit der Anfrageoptimierung bedeutet dies, dass der Nutzer bereits vor der eigentlichen Optimierung seine Interessen definiert und in Form der Gewichte beschrieben haben muss.

Prinzipiell handelt es sich bei den durch die gewichtete Summe identifizierten Optima um eine Teilmenge der Pareto-Menge (siehe Abschnitt 4.2.3.2). So existieren zum Beispiel Suchverfahren für die Pareto-Menge, welche auf der gewichteten Summe basieren (die Pareto-Menge kann zum Beispiel durch ein Gradientenabstiegsverfahren über den Gewichten ermittelt werden). Aufgrund der Tatsache, dass die Gewichte der gewichteten Summe immer eine konvexe Hülle bilden, d.h. es gilt $\sum_i \lambda_i = 1$, finden entsprechende Verfahren lediglich Elemente, welche auf der konvexen Hülle der Pareto-Menge liegen. M.a.W. die Pareto-Menge kann Lösungen enthalten, welche nicht durch das Verfahren der gewichteten Summe gefunden werden, aber dennoch optimal sind und für den Nutzer unter Umständen geeigneter sind als die auf Basis der gewichteten Summe bestimmten Lösungen.

Ein weiteres Problem bei der Überführung eines mehrdimensionalen Optimierungsproblems in eine gewichtete Summe sind verschiedene Wertebereiche der einzelnen Zielfunktionen. Bildet zum Beispiel eine Zielfunktion auf Werte im Intervall $[0, 1]$ ab und eine zweite auf Werte im Intervall $[0, 100]$, so geht der Wert der zweiten Zielfunktion immer um einen Faktor 100 stärker in die gewichtete Summe ein als der erste. Um eine Vergleichbarkeit zu gewährleisten, müssen alle Dimensionen entsprechend normalisiert werden. Sind für eine Zielfunktion der minimale Wert min und der maximal mögliche Wert max bekannt, so kann ein Wert v dieser Zielfunktion auf einfache Weise auf einen Wert im Intervall $[0, 1]$ abgebildet werden: $v' = \frac{v - min}{max - min}$. Sowohl für den Energieverbrauch als auch für die Systemlast der zentralen Komponente lassen sich Minima und Maxima bestimmen, so dass eine entsprechende Normalisierung möglich ist.

Neben der einfachen gewichteten Summe existieren noch weitere Ansätze (Maximum-Ansatz, lexikographische Ordnung der einzelnen Dimensionen, komplexe Gewichtungsansätze) zur Überführung eines multi-kriteriellen Problems in ein einkriterielles Problem. Da diese entweder ein einzelnes Optimierungsziel zu stark in den Vordergrund

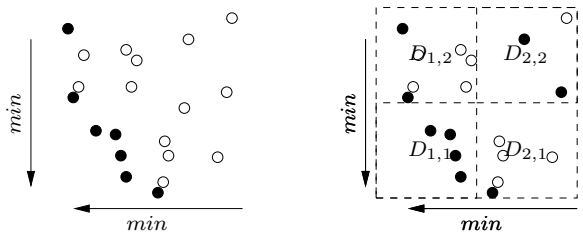


Abbildung 4.7: Pareto-Optimierung: (a) Pareto-Menge, (b) *Divide & Conquer* nach [38]

stellen oder ein hohes Maß an A-Priori-Wissen verlangen, scheinen sie für den Einsatz zur Anfrageoptimierung nur bedingt geeignet.

4.2.3.2 Pareto-Optimierung

Unter den Pareto-Optima bzw. der Pareto-Menge bezüglich eines gegebenen Optimierungsproblems versteht man die Menge der Lösungen, welche nicht von anderen Lösungen dominiert werden. Genauer heißt dies, dass für keine der Lösungen in dieser Menge eine andere Lösung existiert, welche bezüglich aller Optimierungsziele gleich gut und in Bezug auf mindestens ein Optimierungsziel besser ist. Letztendlich muss der Nutzer auf Basis dieser Menge entscheiden, welche Lösung für ihn die günstigste ist. Im Fall der Anfrageoptimierung bedeutet dies, dass ein Anfrageplan $\bar{q}_i$ genau dann in der Pareto-Menge ist, wenn gilt:

$$\bar{q}_i \succ \bar{q}_j \Leftrightarrow \forall \bar{q}_i \neq \bar{q}_j : \mathsf{E}(\bar{q}_i) \leq \mathsf{E}(\bar{q}_j) \wedge \mathsf{C}(\bar{q}_i) \leq \mathsf{C}(\bar{q}_j)$$

Abbildung 4.7(a) zeigt eine Beispielmenge, wobei die dunklen Punkte die Pareto-Menge (oder auch Pareto-Front) beschreiben, aus denen der Nutzer letztendlich sein Optimum wählt.

Das Problem der Pareto-Optimierung ist auch als Maximum-Vektor-Problem [175] bzw. im Kontext von Datenbanksystemen als Skyline-Problem [38] bekannt. Zum Finden der Pareto-Menge entstand eine Vielzahl von Verfahren. Ein trivialer Ansatz ergibt sich dabei durch das Vergleichen jedes einzelnen Elementes mit jedem anderen Element der Ausgangsmenge. Die Autoren in [38] präsentierten mit dem *Block-Nested-Loop*-Verfahren und einem *Divide-and-Conquer*-Ansatz zwei Verfahren, welche mit beschränktem Speicherbedarf zurechtkommen. Beiden Verfahren liegt die Idee zugrunde, Elemente innerhalb von Wertebereichsblöcken miteinander zu vergleichen. Werden in einem Block Elemente durch andere dominiert, so müssen diese im Weiteren nicht betrachtet werden und können aus dem Suchraum entfernt werden. Das *Block-Nested-Loop*-Verfahren vergleicht dabei sequentiell alle Elemente der Ausgangsmenge (analog

zum trivialen Ansatz). Beim Divide-and-Conquer-Verfahren wird die Ausgangsmenge rekursiv in immer kleinere Teilmengen geteilt. Für diese werden getrennte Pareto-Mengen bestimmt. Anschließend werden die resultierenden Pareto-Teilmengen so lange miteinander verschmolzen, bis alle Teilmengen zur Pareto-Menge der Ausgangsmenge vereint sind (siehe Abbildung 4.7(b)).

Weitere interessante Lösungen sind der *Nearest-Neighbor*-Ansatz nach [132] und das *Branch-and-Bound*-Skyline-Verfahren [171]. Die Besonderheit dieser Verfahren ist die stufenweise Berechnung der Ergebnisse, so dass bereits nach kurzer Zeit erste Teile der Skyline vorliegen. Im Kontext obiger Anfrageoptimierung bedeutet dies, dass der Nutzer, insofern ihm die bereits gelieferten Ergebnisse ausreichend sind, bereits nach kurzer Zeit entscheiden kann, welcher Plan materialisiert werden soll. Genügt keines der bisher gelieferten Ergebnisse seinen Anforderungen, so muss er u.U. auf das vollständige Ergebnis warten. Einen umfangreichen Überblick über Arbeiten im Bereich der Pareto-Optimierung liefert [89].

4.2.3.3 Vom Pareto-Optimum zur gewichteten Summe

Die in den letzten beiden Abschnitten präsentierten Verfahren weisen beide jeweils Vor- und Nachteile auf. Lösungen für das Problem der gewichteten Summe sind mit relativ geringem Aufwand bestimmbar, die Definition der Gewichte erfordert aber Expertenwissen. Im Gegensatz dazu ist die Berechnung der Pareto-Menge mit erheblichen Kosten verbunden, welche im Allgemeinen immer über denen für die Berechnung der gewichteten Summe liegen. Da die Pareto-Menge allerdings alle Optima für das gestellte Problem beinhaltet, kann der Nutzer anhand der Ergebnisse entscheiden, welche Lösung seinen Forderungen am besten genügt. Eine Festlegung im Voraus ist nicht notwendig.

Dies ist einerseits wünschenswert, birgt aber auch Nachteile:

- Der Nutzer muss bei jeder neuen Anfrage aus der ermittelten Pareto-Menge erneut den von ihm präferierten Plan auswählen.
- Eine adaptive Reoptimierung laufender Anfragen (siehe Abschnitt 4.4) erfordert u.a. eine erneute Bewertung durch den Nutzer. Erfolgt die Bewertung auf Basis der Pareto-Menge, so ist bei jeder Adaption ein manueller Eingriff durch den Nutzer erforderlich. Dies ist bei häufigen Adaptionen nicht erwünscht.

Im Folgenden soll ein Ansatz beschrieben werden, welcher in einem ersten Schritt die Pareto-Menge $\mathcal{P}$ über allen physischen Plänen für eine Anfrage bestimmt. Aus dieser wählt der Nutzer anschließend den von ihm präferierten und zu realisierenden Anfrageplan $\bar{q}_U$ aus. Auf Basis der Pareto-Menge und der Nutzerentscheidung bestimmt das System dann Gewichte, welche eine automatische Optimierung späterer Anfragen bzw. eine automatische Reoptimierung laufender Anfragen ermöglicht.

Im Folgenden soll sich o.B.d.A. auf die Minimierung der Optimierungskriterien als Optimierungsziel beschränkt werden. Die Überführung in ein Maximierungsproblem ist,

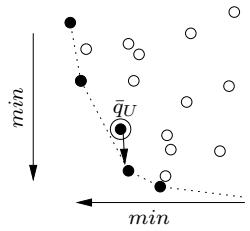


Abbildung 4.8: Transformation von $\bar{q}_U$ auf die konvexe Hülle

wie bereits erwähnt, mittels Negation auf einfache Weise möglich. Im Weiteren soll gezeigt werden, wie die entsprechenden Gewichte auf Basis der Pareto-Menge und der Nutzerentscheidung $\bar{q}_U$ bestimmt werden.

Für den vom Nutzer gewählten Anfrageplan $\bar{q}_U$ muss gelten, dass die auf Basis der Gewichte bestimmte Summe kleiner oder gleich der gewichteten Summe der anderen Elemente der Pareto-Menge ist (es muss von dem verwendeten Optimierungsverfahren eindeutig als Minimum identifiziert werden bzw. mindestens genauso gut sein wie alle anderen Lösungen). Für $\bar{q}_U$ gilt also:

$$\forall_{\bar{q}_i \in \mathcal{P} \setminus \bar{q}_U} : \lambda_1 \mathsf{E}(\bar{q}_U) + \lambda_2 \mathsf{C}(\bar{q}_U) \leq \lambda_1 \mathsf{E}(\bar{q}_i) + \lambda_2 \mathsf{C}(\bar{q}_i)$$

Nach einer entsprechenden Transformation unter Verwendung der Bedingung $\lambda_1 + \lambda_2 = 1$ mit $\lambda_i \geq 0$ ergibt sich für $\mathsf{E}(\bar{q}_U) - \mathsf{C}(\bar{q}_U) - \mathsf{E}(\bar{q}_i) + \mathsf{C}(\bar{q}_i) > 0$

$$\lambda_1 \leq \frac{\mathsf{C}(\bar{q}_i) - \mathsf{C}(\bar{q}_U)}{(\mathsf{E}(\bar{q}_U) - \mathsf{E}(\bar{q}_i)) + (\mathsf{C}(\bar{q}_i) - \mathsf{C}(\bar{q}_U))}$$

bzw. für $\mathsf{E}(\bar{q}_U) - \mathsf{C}(\bar{q}_U) - \mathsf{E}(\bar{q}_i) + \mathsf{C}(\bar{q}_i) < 0$

$$\lambda_1 \geq \frac{\mathsf{C}(\bar{q}_i) - \mathsf{C}(\bar{q}_U)}{(\mathsf{E}(\bar{q}_U) - \mathsf{E}(\bar{q}_i)) + (\mathsf{C}(\bar{q}_i) - \mathsf{C}(\bar{q}_U))}$$

λ_1 kann hierbei ein Intervall überdecken. Jeder Wert aus diesem minimiert die Zielfunktionen wie gefordert. Im Weiteren wird für λ_1 durch das System ein Wert aus dem Intervall bestimmt. Basierend auf λ_1 kann im Anschluß λ_2 bestimmt werden. Da die Elemente der Pareto-Menge die restlichen Elemente dominieren, ist es ausreichend, auf Basis dieser die Gewichte zu bestimmen.

Prinzipiell handelt es sich bei der Pareto-Menge um eine nicht-konvexe Menge von Elementen. Wählt der Nutzer als optimalen Anfrageplan $\bar{q}_U$ ein Element, welches nicht auf der konvexen Hülle der Pareto-Menge liegt, so können aufgrund der Konvexkombination keine Gewichte bestimmt werden, die bei einer Optimierung auf Basis der gewichteten Summe $\bar{q}_U$ bestimmen. Eine einfache Lösung für dieses Problem ist die Transformation des Nutzerwunsches auf Elemente der konvexen Hülle. Hierzu wird für den vom Nutzer bestimmten Plan $\bar{q}_U$ der euklidische Abstand zu allen Plänen $\bar{q}_i$ der konvexen Hülle der Pareto-Menge bestimmt und der Plan mit dem geringsten Abstand zu $\bar{q}_U$ für die Berechnung der Gewichte herangezogen (siehe Abbildung 4.8).

4.3 Statische Anfrageoptimierung

Nachdem im vorangegangen Abschnitt die Kostenbewertung von Operator-Graphen beschrieben wurde, soll in den folgenden Abschnitten die initiale Anfrageoptimierung in *AnduIN* als Ganzes betrachtet werden. Diese entspricht dabei den in Abbildung 4.1(b) grau hinterlegten Schritten: algebraische Optimierung, physische Optimierung und abschließende Plan-Dekomposition.

Zumeist wird die logische Anfrageoptimierung als Transformation mittels einer Menge von Rewriting-Regeln umgesetzt [91]. In Abschnitt 4.3.1 soll auf das Anfragerewriting in *AnduIN* eingegangen werden. Anschließend wird in Abschnitt 4.3.2 auf die umgesetzte Planenumeration als Teil der physischen Optimierung eingegangen. Schwerpunkt sind hierbei die Integration komplexer Operatoren in den Optimierungsprozess und die Verwendung Suchraum beschränkender Heuristiken bei der Planenumeration.

4.3.1 Rewriting

Das *rewriting* beschreibt die Umformung von Anfrageplänen auf Grundlage von Regelmengen, welches an zwei Stellen innerhalb der Anfrageverarbeitung Verwendung findet. Während der logischen bzw. algebraischen Optimierung wird der durch die Sichtexpansion erzeugte Parse-Plan auf Basis einer Menge von Äquivalenzregeln für relationale Operatoren und verschiedener Heuristiken umgeformt. Ziel dieser logischen Transformation ist eine geeignete Vorauswahl für die anschließende physische Optimierung.

Die folgenden Heuristiken sind in gängigen DBMS-Optimierern als Standard implementiert und wurden in dieser Form im Anfrageoptimierer der zentralen Komponente von *AnduIN* realisiert [204]:

- Selektionen und Projektionen werden so weit wie möglich in Richtung Datenquellen verschoben. Das Verschieben dient der frühzeitigen Reduktion der Datenmenge.
- Sofern möglich, werden Selektionen und kartesisches Produkt zu einem EquiJoin verbunden. Die Größe der Ergebnismenge eines Verbundes liegt normalerweise deutlich unter der des Produktes.
- Duplikate werden eliminiert.

Die anschließende Transformation des logischen Anfrageplanes in eine Menge äquivalenter physischer Ausführungspläne kann ebenfalls als eine Form des Rewriting verstanden werden. Die Regeln bilden dabei einen logischen Operator auf eine Menge von physischen Operatoren ab. Im Weiteren werden zusätzlich benötigte Regeln, welche für die Verarbeitung von Synopsen-Operatoren und das Sampling notwendig sind, genauer beschrieben.

4.3.1.1 Komplexe Operatoren

Enthält eine Anfrage die Definition eines oder mehrerer komplexe Verfahren, so werden diese in entsprechende Synopsen-Operatoren ξ transformiert. Die Parameter des Verfahrens werden dabei dem Operator als Parameter mitgegeben sowie die Attribute über denen die Analyse erfolgen soll.

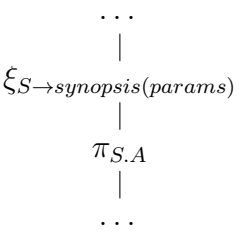
Aufgrund der attribut-übergreifenden Analyse von Synopsen-Operatoren (alle Attribute sind in die Analyse eingeschlossen) können Operatoren wie die Projektion und die Selektion *nicht* über einen entsprechenden Synopsen-Operator hinweg in Richtung Quellen verschoben werden. Dies könnte die Charakteristik der zu analysierenden Daten verändern und somit die Ergebnismenge des Synopsen-Operators ändern.

Im Folgenden sollen einige Beispiele die Transformation einer Anfrage mit komplexen Operatoren verdeutlichen.

Beispiel 4.5 Es werden zwei Datenströme $S_{A,B}$ und $R_{X,Y}$ angenommen. A, B bzw. X, Y bezeichnen dabei die Attribute der jeweiligen Datenströme. Das erste Beispiel zeigt eine einfache Anfrage mit einer zusätzlichen Projektion. Diese wird in nebenstehenden Operator-Graphen übersetzt:

```
SELECT A
FROM S [ synopsis (params) ]
...
```

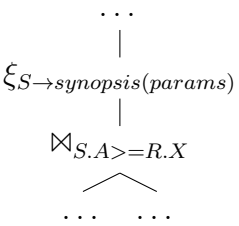
$\Rightarrow$



Im zweiten Beispiel wird die Analyse bei vorherigem Verbund zweier Datenströme gezeigt. Die Analyse bezieht sich dabei lediglich auf den Datenstrom s.

```
SELECT *
FROM S [ synopsis(params) ], R
WHERE S.A >= R.X
```

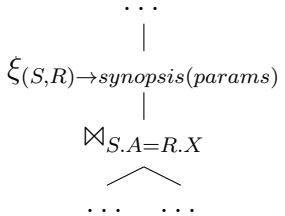
$\Rightarrow$



Eine komplexe Analyse über zwei Datenströme hinweg zeigt das nächste Beispiel.

```
SELECT *
FROM
   (SELECT *
    FROM S,R
    WHERE S.A = R.X)
    [ synopsis(param) ]
...
```

$\Rightarrow$



■

4.3.1.2 Projektion auf Sampling

Eine Besonderheit bei der Verarbeitung von Datenströmen mit dem Hintergrund der WSNs stellt der Sampling-Operator für das Generieren von Daten auf den Sensorknoten dar. Im Gegensatz zu „normalen“ Datenstromquellen, deren Schema zum Anfragezeitpunkt fest steht (und dem System mit der Registrierung eines Stromes bekannt gegeben wird), kann es beim Sampling von physischen Werten sinnvoll sein, das Schema während der Anfrageoptimierung anzupassen.

Neben dem Versenden von Daten trägt insbesondere die Datenerfassung auf den Sensorknoten mit einem hohen Energiebedarf zum Verbrauch bei. Es erscheint daher sinnvoll, Daten, welche im weiteren Verlauf der Anfragen nicht benötigt werden, durch eine Projektion von der Erfassung auszuschließen. Eine einfache Möglichkeit, dies zu tun, ist das Zusammenführen des Sampling-Operators ζ mit einer direkt folgenden Projektion π. Zu diesem Zweck wurde die Regel

$$\pi_{R \subseteq S}(\zeta_S) \rightarrow \zeta_R$$

der Regelmenge hinzugefügt. S bezeichnet dabei das vom Sampling-Operator erzeugte Schema und $\{A_1, \ldots, A_k\} = R \subseteq S$ die Menge an Attributen, die nach der Projektion im Datenstrom verbleiben.

4.3.2 Planenumeration

Während der Enumerationsphase wird der logische (algebraisch) optimierte Anfrageplan in eine Menge äquivalenter physischer Ausführungspläne übersetzt. Der einfachste Ansatz hierbei ist die erschöpfende Ersetzung. Bei dieser werden alle möglichen (sinnvollen) Kombinationen von Operatoren (zum Beispiel Verbundreihenfolgen) und alle möglichen physischen Implementierungen für einen Operator zu physischen Ausführungsplänen zusammengesetzt. Da die erschöpfende Ersetzung aufgrund der Vielzahl von resultierenden physischen Plänen mit einem hohen Kostenaufwand verbunden ist, wurden im Kontext der DBMS verschiedene Heuristiken für die Planenumeration entwickelt. Prinzipiell lassen sich die Übersetzungsstrategien in zwei Klassen einordnen:

- *Top-Down*-Strategien: Von der Wurzel beginnend werden zu den Datenquellen hin logische Operatoren durch ihre physischen Implementierungen ersetzt. Üblicherweise enthält der logische Anfrageplan kommutative und assoziative Gruppen, welche während der Enumerationsphase aufgelöst werden müssen. Das Auflösen dieser und die u.U. große Anzahl möglicher physischer Implementierungen für einen Operator können zu einer großen Menge an alternativen physischen Plänen führen.

- *Bottom-Up*-Strategien: Die Transformation erfolgt von den Quellen beginnend. Die aktuelle Wurzel des betrachteten Teilplanes wird durch die möglichen physischen Implementierungen ersetzt. Üblicherweise werden während der Konstruktionsphase gleich die Kosten für den Teilplan ermittelt. Unter der Annahme, dass erzeugte Teilpläne zu einem späteren Zeitpunkt nicht besser werden, werden Pläne mit hohen Kosten sofort eliminiert.

Im Kontext von *AnduIN* bietet sich die *Bottom-Up*-Strategie an, da sich hier leicht einfache Heuristiken zur Reduktion der Planmenge integrieren lassen. In der Literatur finden sich verschiedene mehr oder weniger praxisnahe Lösungen [204]:

- *Heuristic Selection*: Basierend auf Erfahrungswerten werden Regeln für das Übersetzen von logischen Plänen aufgestellt. Die Regeln nehmen dabei sowohl auf die Reihenfolge der Übersetzung als auch auf konkrete Implementierungen Bezug.

- *Branch-and-Bound*: Es wird ein beliebiger (guter) Plan bestimmt. Anschließend werden Teilbäume des Planes auf Alternativen hin überprüft. Sind die Kosten für den Teilbaum größer als die für den aktuellen Plan, dann wird der Teilbaum verworfen. Sind die Kosten geringer, dann wird der Teilbaum im aktuellen Plan durch den so ermittelten Teilbaum ersetzt. Der Vorteil dieses Verfahrens liegt in der sukzessiven Verbesserung des Planes. Die Enumeration kann so zu einem beliebigen Zeitpunkt abgebrochen werden und der aktuell beste Plan wird als Ergebnis zurückgeliefert.

- *Hill Climbing*: Der logische Plan wird auf Basis des heuristischen Ansatzes (Heuristic Selection) übersetzt. Anschließend werden mit dem Ziel der Verbesserung kleine Änderungen an diesem Plan vorgenommen (Reihenfolge- oder Implementierungsänderungen einzelner Operatoren).

- *Dynamic Programming:* Für jeden Teilausdruck (der Bottom-Up-Strategie) wird lediglich die aus Kostensicht beste Alternative aufgehoben. Gruppierungen und assoziative Gruppen werden erst zur Laufzeit auf Basis einer gesonderten Kostenanalyse ausgewertet.

Da die Anfrageoptimierung auf der zentralen Instanz von *AnduIN* stattfindet, spielt die Performance der Enumeration nur eine untergeordnete Rolle. Aus diesem Grund wurde ein einfacher heuristischer Ansatz umgesetzt. Der logische Anfrageplan wird auf Grundlage einer Regelmenge in eine Menge von physischen Anfrageplänen übersetzt. *Nach* der Enumerationsphase werden die physischen Pläne auf Basis des in Abschnitt 4.2 beschriebenen Kostenmodells bewertet.

Im Folgenden soll ein alternativer Ansatz beschrieben werden, welcher physische Pläne während der Konstruktionsphase bewertet und Mengen von nicht effizienten alternativen Plänen bereits vor der vollständigen Erzeugung verwirft (ähnlich dem Branch-and-Bound-Ansatz).

4.3.2.1 Iterative multi-kriterielle Planenumertion

Handelt es sich beim Optimierungsziel um ein eindimensionales Problem, dann ist es oftmals ausreichend, einen einfachen Greedy-Ansatz zu verfolgen. Hierbei wird während jedes Iterationsschritts der Konstruktionsphase der Operator gewählt, welcher die voraussichtlichen Kosten minimiert. Im vorliegenden Fall der multi-kriteriellen Optimierung ist das sukzessive Verwerfen von Plänen während der Bottom-Up-Strategie jedoch nicht möglich. So kann es zum Beispiel vorkommen, dass bereits mehrere Operatoren des physischen Anfrageplanes integriert wurden, aber bisher lediglich Kosten für eine Dimension existieren (die Kosten der anderen Dimensionen sind währenddessen noch undefiniert). Im Beispiel der vorliegenden Arbeit bedeutet dies konkret, dass die Energiekosten für den INQP-Teil bereits feststehen können und daher mit denen anderer äquivalenter Pläne verglichen werden können, aber noch keine Aussage über die voraussichtlichen Ausführungskosten auf der DSMS-Komponente möglich sind. Daraus lassen sich zwei Probleme ableiten: (i) Die bisher nicht berücksichtigten Dimensionen würden immer mit Kosten von 0 in die Bewertung eingehen und (ii) im Falle eines zweidimensionalen Optimierungsproblems wie dem betrachteten würde das Problem somit auf ein eindimensionales Problem reduziert werden. Im Fall der Energie- vs. Ausführungskosten-Minimierung würde somit immer der Plan mit dem geringsten Energieverbrauch alle anderen dominieren. Da die Anwendung des Greedy-Verfahrens auf das multi-kriterielle Optimierungsproblem nicht möglich ist, soll im Folgenden ein alternativer Ansatz beschrieben werden.

In einem ersten Schritt wird ein beliebiger physischer Plan vollständig aus dem logischen Plan abgeleitet (*deep-first approach*) und der Pareto-Menge hinzugefügt. Anschließend werden zufällig alternative physische Pläne entwickelt und mit den Plänen der aktuellen Pareto-Menge auf Dominanz hin verglichen. Wird ein so erzeugter Plan nicht durch Pläne der Pareto-Menge dominiert, so wird er dieser hinzugefügt. Die Bewertung des Planes und der Vergleich mit der Pareto-Menge erfolgt während der Konstruktion des Planes nach dem Hinzufügen jedes einzelnen Operators. Verursacht ein so erzeugter Teilplan in *allen* Dimensionen größere Kosten als die Pläne, welche zu diesem Zeitpunkt Teil der Pareto-Menge sind, so kann dieser Plan nicht mehr Teil der Pareto-Front werden. Des Weiteren können alle Pläne, die sich aus diesem Plan durch Hinzufügen weiterer physischer Operatoren ableiten lassen, aus der Menge der zu prüfenden Anfragepläne entfernt werden. Da ein weiteres Hinzufügen von Operatoren die Kosten nur noch weiter erhöhen kann, ist deren Berücksichtigung nicht mehr notwendig.

4.3.2.2 Suchraumbeschränkende Heuristiken

Eines der größten Probleme bei der Planenumeration ist die Größe des aufgespannten Suchraumes. Im Weiteren werden in *AnduIN* umgesetzte Heuristiken vorgestellt, welche die Menge der zu erzeugenden physischen Pläne bereits während der Konstruktionsphase verringern können:

INQP-DSMS-Übergang Teilergebnisse, welche einmal an die Basisstation gesendet wurden, werden nicht erneut in das WSN eingebracht (eine Ausnahme bildet die Feedback-Schleife bei entsprechenden komplexen Operatoren). Ein logischer Plan q wird von der Wurzel beginnend in Post-Order Reihenfolge-übersetzt, d.h. für jeden logischen Operator op_i werden alle möglichen physischen Operatoren $\bar{op}_i^k \in \bar{\mathbf{O}}_i$ eingesetzt. Für das Übersetzen von Operatoren gilt die folgende Einschränkung:

Wird ein logischer Operator op_i in einen physischen Operator $\bar{op}_i \in \bar{\mathbf{O}}_i^{\mathcal{L}}$ übersetzt, so kann die Realisierung des Eltern-Operators op_j von op_i ebenfalls nur aus der Menge der auf der zentralen Instanz verarbeiteten Operatoren gewählt werden, d.h.

$$op_j = pred_q(op_i) \wedge \bar{op}_i \in \bar{\mathbf{O}}_i^{\mathcal{L}} \Rightarrow \bar{op}_j \in \bar{\mathbf{O}}_j^{\mathcal{L}}.$$

Falls noch nicht geschehen, so wird an der Stelle des Übergangs zwischen INQP und DSMS ein Verbindungsmanager eingefügt. Mit der oben beschriebenen Einschränkung gibt es pro Pfad (von einem Wurzelknoten zur Senke) maximal einen Verbindungsmanager. Der Verbindungsmanager wird durch die zwei physischen Operatoren $\overline{conn}^{\mathcal{N}\uparrow}$ (In-Network-Senke) und $\overline{conn}^{\mathcal{L}\downarrow}$ (Quelle auf der zentralen Instanz) repräsentiert. Bei komponenten-übergreifenden komplexen Operatoren kann auf einen extra Verbindungsmanager verzichtet werden, da diese Klasse von Operatoren bereits implizit entsprechende Komponenten aufweist.

Speicherbeschränkungen Aufgrund der Hardwarebeschränkungen der Sensorknoten können beliebige Pläne bzw. Teilpläne nicht immer im WSN ausgeführt werden. So stellt beispielsweise der verfügbare Speicherplatz auf den Sensoren einen beschränkenden Parameter dar. Die für die Evaluierung in Abschnitt 6.2 benutzten Sensoren verfügen zum Beispiel über 512KByte Flashspeicher und 98KByte RAM. Im Folgenden wird davon ausgegangen, dass alle Knoten das gleiche Softwareimage erhalten und dass alle Sensoren über die gleiche Ausstattung an Speicher verfügen.

Der Speicherbedarf der Laufzeitumgebung setzt sich wie folgt zusammen:

$$mem_{img} \leq mem_{img}(os) + \sum_{\bar{q}^{\mathcal{N}} \in EXE} \left(\sum_{\bar{op} \in \bar{q}^{\mathcal{N}}} mem_{img}(\bar{op}) \right)$$

Zusätzlich zum Speicherbedarf des Sensorknoten-Betriebssystems $mem_{img}(os)$ belegt jeder Operator der Laufzeitumgebung eine konstante Menge an Speicher $mem_{img}(\bar{op})$. Die Menge EXE umfasst alle sich gegenwärtig in der Laufzeitumgebung befindenden Anfragen. Sowohl der Speicherbedarf für den Betriebssystemkern als auch für die einzelnen Operatoren ist unabhängig von Laufzeitparametern und kann durch einmaliges Messen bestimmt werden.

Eine Auslagerung von Teilen der Anfrage ist nur möglich, sofern der vorhandene Speicher eines einzelnen Sensorknotens nicht überschritten wird. *AnduIN*-Module, welche immer vorhanden sein müssen, können an dieser Stelle als Teil des ständig belegten

Speichers $mem_{img}(os)$ angesehen werden und müssen damit bei der Abschätzung nicht extra betrachtet werden.

Obige Abschätzung liefert lediglich eine obere Grenze für den tatsächlichen Speicherverbrauch. Im Allgemeinen nehmen Compiler eine Vielzahl an Optimierungen vor, welche u.a. den Speicherverbrauch betreffen.

Auch der Speicherbedarf für Daten im RAM ist beschränkt und muss bei der Planenumeration berücksichtigt werden. Die Abschätzung ist ähnlich der Flash-Speicherabschätzung:

$$mem_{RAM}(\bar{q}) \leq mem_{RAM}(os) + \sum_{\bar{q}^{\mathcal{N}} \in EXE} \left(\sum_{\bar{op} \in \bar{q}^{\mathcal{N}}} mem_{RAM}(\bar{op}) \right) \tag{4.4}$$

Der Speicher für Betriebssystem und statische Laufzeitkomponenten $mem_{RAM}(os)$ wird wiederum als fest angenommen. Im Gegensatz zum Flash-Speicherbedarf ist der RAM-Speicherbedarf der einzelnen Operatoren $mem_{RAM}(\bar{op})$ zur Laufzeit nicht konstant. In Abhängigkeit des verwendeten Verfahrens können wenige Byte bis zu mehreren KByte für notwenige Synopsen anfallen. Für einfache Filter und die Projektion ist die Speicherermittlung wieder nahezu trivial. Bei fensterbasierten Operatoren müssen der für den Operator benötigte (weitestgehend konstante) Speicher und zusätzlich der für die zwischenzuspeichernden Datensätze notwendige Speicher ermittelt werden. Da die Fenstergröße über die Spezifikation in der Anfragebeschreibung zum Zeitpunkt der Anfrageoptimierung bekannt ist, kann der dafür benötigte Speicher einfach berechnet werden (eine entsprechende Abschätzung für fensterbasierte Operatoren findet sich in [43]). Bei Operatoren mit komplexen Synopsen müssen zusätzliche Kosten-Modelle entwickelt werden. Da es sich bei allen eingesetzten Algorithmen um Verfahren für Datenströme handelt, sind diese in ihrem benötigten Speicherbedarf immer nach oben beschränkt (Definition für datenstromgeeignete Verfahren [78]), so dass sich entsprechende Speicher-Obergrenzen identifizieren lassen.

Alle gleichzeitig ausgeführten Anfragen sind Teil der gleichen Laufzeitumgebung (insofern sie über mindestens eine In-Network-Quelle verfügen). Die verschiedenen Anfragen teilen sich somit den zur Verfügung stehenden Speicher (sowohl Programm- als auch Datenspeicher). Das Problem der bestmöglichen Speicherauslastung steht dabei in Konkurrenz zu den sonstigen Optimierungszielen. Prinzipiell lassen sich für eine optimale Speicherauslastung zwei grundlegende Lösungssätze finden:

- *sequentielles Füllen mit Operatoren*: Solange noch freier Speicher in der Laufzeitumgebung zur Verfügung steht, werden Operatoren einer neuen Anfrage in diesen Speicher integriert. Pläne, deren Speicherbedarf, sowohl Flash als auch RAM, über den noch vorhandenen Speicherplatz hinausgeht, müssen bei der Planenumeration nicht weiter betrachtet werden.

- *anfrageübergreifende Speicherbetrachtung*: Wird eine neue Anfrage hinzugefügt, dann werden alle gegenwärtig laufenden Anfragen erneut geprüft und zwar sowohl

hinsichtlich des Gesamtspeicherbedarfs als auch bezüglich der Optimierungziele. Hierfür ist eine erneute vollständige Planenumeration für alle laufenden Anfragen notwendig, was zu einem deutlich höheren Aufwand gegenüber dem ersten Ansatz führt.

Der Ansatz des sequentiellen Füllens ist mit einem deutlich geringeren Entwicklungsaufwand verbunden. Da lediglich neue Anfragen auf ihr Optimum hin untersucht werden müssen, ist auch der Aufwand für die eigentliche Optimierung gegenüber dem zweiten Lösungsansatz deutlich geringer. Demgegenüber wird die Laufzeitumgebung bei der anfrageübergreifenden Speicherbetrachtung optimal genutzt.

4.4 Adaptive Anfrageoptimierung

Schwerpunkt des im Rahmen der Arbeit entwickelten Systems ist die Verarbeitung von Langzeit-Anfragen. Neben der im letzten Abschnitt beschriebenen einmaligen initialen Anfrageoptimierung ergeben sich im Zusammenhang mit kontinuierlichen Anfragen die folgenden Probleme:

- Zum Zeitpunkt der initialen Optimierung der Anfragen sind oftmals keine oder nur unzureichende Statistiken über die verwendeten Datenströme bekannt. Dies kann zu einer ungenügenden oder fehlerhaften Optimierung von Anfragen führen.
- Das Verhalten der Datenströme (zum Beispiel die Verteilung der Daten und deren Ankunftsrate) kann sich während der Ausführungsphase ändern.
- Die vom zugrundeliegenden (Betriebs-) System zur Verfügung gestellten Ressourcen können sich zur Laufzeit ändern. Mögliche Ursachen hierfür können zum Beispiel parallel bearbeitete Anfragen sein oder eine Auslastung, welche unabhängig vom System auftritt (zum Beispiel durch externe Dienstprogramme).

Die sich stetig verändernden Laufzeitparameter machen eine Reoptimierung (Adaption) laufender Anfragen notwendig. Ohne diese kann es zu signifikanten Verschlechterungen der optimierten Eigenschaften kommen.

Für die Reoptimierung sind im Wesentlichen zwei Komponenten notwendig: (i) eine Menge von Monitoren, die laufende Operatoren hinsichtlich bestimmter Charakteristiken überwacht und (ii) eine Komponente, welche die Reoptimierung anstößt und den Planaustausch vornimmt (siehe Abbildung 4.1(b)).

Im Kontext des hier vorgestellten Systems ergeben sich zwei Einsatzgebiete für die Reoptimierung von Anfragen: *lokale Reoptimierung* des DSMS-Teils der Anfrage und eine *globale Reoptimierung* der Gesamtanfrage. Für die Reoptimierung der lokal (von der DSMS-Komponente) ausgeführten Anfrageteile existiert in der Literatur bereits eine Vielzahl von verschiedenen Verfahren. Abschnitt 4.4.1 gibt einen kurzen Überblick über diese.

Anders verhält sich dies bei der globalen Reoptimierung der Gesamtanfrage, welche sowohl den DSMS- als auch den INQP-Teil betrifft. Hierbei ergeben sich im Zusammenhang mit der vorgestellten Idee der Anfrage-basierten Laufzeitumgebung zwei grundlegende Probleme: (i) Änderungen müssen (energie-)kostenintensiv im WSN propagiert werden und (ii) die Laufzeitumgebung umfasst nur den zum Zeitpunkt der initialen Optimierung erstellten Plan. In Abschnitt 4.4.2 sollen verschiedene Möglichkeiten zur Aktualisierung des INQP-Teils von *AnduIN* betrachtet werden. Anschließend werden in Abschnitt 4.4.3 die für die Aktualisierung notwendigen Kosten auf Grundlage des verwendeten Anfragemodells betrachtet.

4.4.1 Verwandte Arbeiten

Bei der adaptiven Anfrageoptimierung versucht man auf sich ändernde Statistiken zu reagieren und Anfragen zur Laufzeit an die neuen Gegebenheiten anzupassen. Zu diesem Zweck werden Anfragen kontinuierlich während ihrer Ausführung überprüft und notwendige Adaptionen vorgenommen. Aufgrund der prinzipiellen Unabhängigkeit von initialen Statistiken wird die adaptive Anfrageoptimierung oftmals auch zum *autonomic computing* hinzu gezählt. In den letzten Jahren entstand hierzu eine Vielzahl neuer Ansätze sowohl im Rahmen von DBMS als auch DSMS. Die Reoptimierung einer Anfrage zur Laufzeit gliedert sich im Wesentlichen in drei Schritte [65, 113]:

1. Auswahl des Zeitpunktes, zu dem optimiert werden soll,
2. Suche nach dem neuen besten Ausführungsplan, welcher den gegenwärtigen Ausführungsplan ersetzen soll und
3. Austausch des aktuellen Planes durch den neuen Plan.

Jede dieser drei Phasen ist mit zusätzlichen Kosten für das ausführende System verbunden. Das Bestimmen der besten Austauschzeitpunkte ist daher eine der wesentlichen Herausforderungen bei der Reoptimierung.

Eine der ersten Arbeiten, die sich mit dem Thema der adaptiven Optimierung von Anfragen beschäftigte, war [120]. Die Autoren betrachteten dabei komplexe Anfragen in DBMS, wobei sie aber bereits die wesentlichen Herausforderungen für die Adaption von Anfragen in DSMS formulierten: die Verteilung der zur Verfügung stehenden Ressourcen und die Reorganisation der Anfrage. Der präsentierte Ansatz basiert auf der Idee, Alternativpläne zu entwickeln, welche im Fall einer Kostenreduktion gegen den laufenden Anfrageplan ausgetauscht werden. Die Autoren betrachten dabei im Rahmen der Optimierung auch bereits anfallende Kosten, welche u.a. durch die Materialisierung von Zwischenergebnissen anfallen. Ein ähnliches Problem wurde in [60] betrachtet. Schwerpunkt hierbei war die autonome Adaption des von den Operatoren belegten Speichers zur Laufzeit. Die Analyse erfolgte dabei auf Basis eines relationalen DBMS.

Mit dem Aufkommen der Datenstromverarbeitung mussten zusätzlich zur Systemauslastung auch Eigenschaften wie zum Beispiel sich ändernde Datencharakteristika bei der

Reoptimierung berücksichtigt werden. In Verbindung mit STREAM wurden bereits erste Verfahren zur adaptiven Optimierung präsentiert. In [25] stellen Babu et al. verschiedene Greedy-Verfahren vor, bei denen die Anpassung der Reihenfolge einfacher Filter- und Verbundoperatoren im Mittelpunkt steht. Der präsentierte A-Greedy-Optimierer besteht dabei im Wesentlichen aus zwei Komponenten: einem *Profiler* und dem eigentlichen *Reoptimierer*. Der Profiler ist für das Sammeln und Verwalten von Statistiken über den einzelnen Datenströmen verantwortlich. Der Reoptimierer entscheidet anschließend anhand dieser (basierend auf einer Invariante), ob die ausgeführten Anfragen angepasst werden müssen.

Bei dem von der Universität Berkeley entwickelten TelegraphCQ wurde bereits bei der Konzeption des Systems auf das Problem der kontinuierlichen Anfrageverarbeitung eingegangen. Die Verwendung von Eddies und SteMs (*State Module*) erlaubt den vollständigen Verzicht auf eine Pre-Optimierung [19]. Die Entscheidung darüber, welche physische Implementierung eines Operators verwendet wird, geschieht durch die SteMs zur Laufzeit. In [177] stellen Raman et al. eine Technik vor, bei welcher der Anwender Teilergebnisse sieht und anschließend Tupel manuell priorisieren kann und somit das Routing der Eddies beeinflusst. Der Eddie-Ansatz hat allerdings auch zwei wesentliche Nachteile: (i) Die modulare Verarbeitung erzeugt einen nicht zu vernachlässigenden Overhead und (ii) die Optimierung findet stets im Hinblick auf nur einen Operator statt. Anfragen als Ganzes werden nicht optimiert.

Einen anderen Ansatz verfolgen die Autoren in [22]. Ziel dieses Verfahrens ist die Verteilung der zur Verfügung stehenden Ressourcen in Abhängigkeit von der momentanen Auslastung. Kommt es zum Beispiel an der Input-Queue eines Operators zu Warteschlangen, so bekommt dieser Operator zur Bearbeitung mehr Rechenzeit zugewiesen. Auf diese Weise kann der Operator die eingehenden Tupel schneller verarbeiten und die Blockierung an diesem Operator wird beseitigt. Neben der Auslastung durch die Anfrageverarbeitung ist auch der Speicherverbrauch für auszuführende Anfragen von Interesse. Eines der wesentlichen Konzepte der Datenstrom-Verarbeitung ist die In-Time-Verarbeitung. Daten können also nicht auf Externspeicher ausgelagert werden. Parallele Anfragen müssen sich somit die zur Verfügung stehenden Ressourcen teilen. Einen entsprechenden Ansatz zur Optimierung der Speicherauslastung präsentieren die Autoren in [20]. Der vorgestellte *Chain-Scheduling*-Ansatz konzentriert sich dabei wiederum auf die Reorganisation von Anfragen über den einfachen Operatoren Filter, Projektion und Verbund.

Ein weiterer interessanter Ansatz zur Anfrageadaption wurde in [43] beschrieben. Die Eigenschaften der ausgeführten Operatoren, zum Beispiel Fenstergrößen und Zeitgranularität, werden dabei in Abhängigkeit der zur Verfügung stehenden Ressourcen gezielt angepasst. Der Ausführungsplan selbst wird nicht verändert. Zum Zweck der Optimierung definiert der Nutzer Grenzwerte, welche dem System die kontinuierliche Anpassung mit dem Ziel einer gleichzeitig maximalen Anfragegüte ermöglichen.

Bereits dieser Überblick zeigt die vielfältigen Möglichkeiten, welche sich für die Reoptimierung von Anfragen in einem datenstrom-verarbeitenden System ergeben. Entsprechend umfangreich sind auch die Lösungsansätze, welche in den letzten Jahren

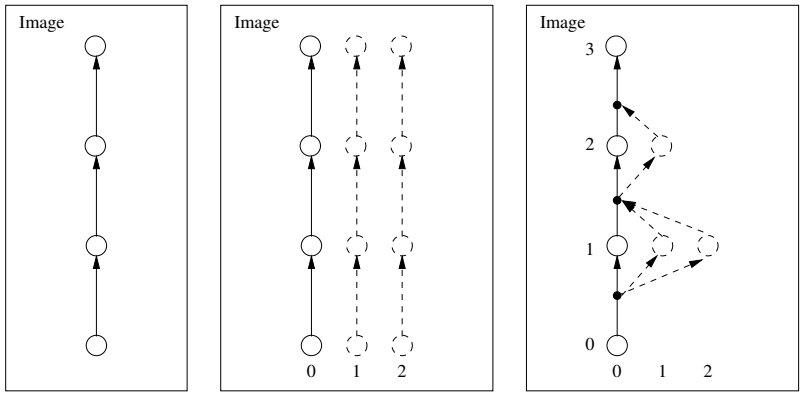


Abbildung 4.9: Integration von Alternativplänen in die INQP-Laufzeitumgebung: (a) einfacher Plan, (b) alternative Pläne, (c) alternative Operatoren

präsentiert wurden. Einen sehr guten Überblick über Arbeiten in diesem Bereich liefern die Arbeiten [24, 66].

Aufgrund der großen Anzahl an Vorarbeiten soll im Weiteren lediglich auf einige spezielle Probleme im Zusammenhang mit der Reoptimierung von Anfragen in *AnduIN* und deren Lösung eingegangen werden.

4.4.2 Reoptimierung des INQP-Teils

Eine der zentralen Komponenten von *AnduIN* ist die INQP-Laufzeitumgebung. Wie in Abschnitt 3.2.2 beschrieben, besteht diese im Wesentlichen aus dem Sensorknoten-Betriebssystem, dem INQP-Koordinator, den eigentlichen Anfragen und möglicherweise benötigten funktionalen Modulen. Erst dieser Ansatz ermöglicht den Einsatz nahezu beliebiger Operatoren im WSN. Mit diesem Ansatz ergeben sich allerdings auch Probleme. Insbesondere die Reoptimierung der Laufzeitumgebung kann mit einem großen Kostenaufwand verbunden sein.

In diesem Abschnitt sollen zwei Ansätze vorgestellt werden, welche in einem begrenzten Rahmen Anpassungen am INQP von *AnduIN* ermöglichen, ohne die Laufzeitumgebung auszutauschen. Ist dies nicht möglich, muss eine Aktualisierung der Laufzeitumgebung (zum Beispiel durch OTA-Programmierung) vorgenommen werden.

Im Weiteren wird davon ausgegangen, dass der von den Sensorknoten zur Verfügung gestellte Speicher für Anwendungen größer ist als der gegenwärtig für die Laufzeitumgebung benötigte. Es können somit weitere alternative Ausführungspläne bzw. alternative Operatoren in das Sensorknotenimage integriert werden. Zwischen diesen kann die zentrale Instanz je nach Bedarf wechseln und so Anpassungen am INQP-Teil einer Anfrage vornehmen ohne ein vollständiges Update zu initialisieren.

Für die Integration zusätzlicher Anfrageobjekte ergeben sich prinzipiell zwei Ansätze:

- *alternative Pläne:* Es werden zum Zeitpunkt der initialen Optimierung zusätzlich zum optimalen Ausführungsplan weitere komplette Ausführungspläne in die Laufzeitumgebung integriert (auch wenn sich diese lediglich durch die Implementierung eines Operators unterscheiden). Unter der Annahme, dass jeder Alternativplan ebenso viel Speicher belegt wie der Originalplan, führt dieser Ansatz zu einer entsprechenden Vervielfachung des Speicherbedarfs pro Anfrage, was wiederum die Menge der Alternativen pro Anfrage einschränkt. Andererseits gestaltet sich der Wechsel auf einen Alternativplan denkbar einfach. Der Optimierer muss den Sensorknoten lediglich mitteilen, auf welchen Plan gewechselt werden soll. Tupel, die sich gegenwärtig noch im alten Plan befinden, laufen diesen bis zum Ende durch. Neue Tupel werden durch den neuen Plan geroutet. Befinden sich Synopsen-Operatoren im Ausführungsplan, dann müssen beide Pläne zeitweise parallel ausgeführt werden. Weist der neue Plan den gleichen Zustand auf wie der alte Plan, dann kann die Ausführung des alten Planes beendet werden.

- *alternative Operatoren:* Wenn möglich, werden für einen Operator alternative Implementierungen in die INQP-Laufzeitumgebung integriert. Im Gegensatz zum Hinzufügen alternativer Pläne ist diese Lösung weniger speicherintensiv, da lediglich einzelne Operatoren in verschiedenen Implementierungen vorhanden sind. Allerdings ist der Wechsel auf einen Alternativplan aufwendiger. Für die Änderung eines Planes benötigt jeder Sensorknoten Informationen darüber, welche Operatoren ausgetauscht werden sollen und welche Implementierungen diese ersetzen. Dies setzt entsprechende Nachrichten vom Optimierer an die Knoten im WSN voraus. Zudem muss zwischen zwei Knoten im Plan jeweils ein einfacher Koordinator eingefügt werden, welcher das Routing zwischen den einzelnen Operatoren übernimmt. Die Integration der Koordinatoren wiederum ist mit zusätzlichen Speicher- und Energiekosten verbunden.

Abbildung 4.9 zeigt die drei INQP-Fälle: ohne zusätzliche Anfrageobjekte (a), mit Alternativplänen (b) und alternativen Operatoren (c).

Prinzipiell ist auch noch eine dritte Variante, die Kombination aus beiden Strategien, möglich. Hierbei werden alternative Teilpläne in die Laufzeitumgebung integriert. Das Verfahren vereint dabei im Wesentlichen die Vorteile der beiden anderen Strategien. Allerdings ist das Problem des Auffindens alternativer Teilpläne entsprechend komplex. Es müssen alle Alternativen vom einelementigen Austausch bis zum Austausch des vollständigen Planes untersucht werden.

Die Entscheidung, welche Alternativen in die Laufzeitumgebung integriert werden, muss vom Optimierer zum Zeitpunkt der Anfrageinitialisierung gefällt werden. Die Laufzeitumgebung wird dabei so lange mit Anfragen aufgefüllt, bis die Speicher-Ressourcen erschöpft sind. Alternative Pläne bzw. alternative Operatoren sollten allerdings nur dann integriert werden, wenn sie bei einem Planwechsel auch eine entsprechende Kostenverbesserung versprechen.

Die gegenwärtige Implementierung von *AnduIN* beschränkt sich auf die Integration des

initial bestimmten Anfrageplanes und sieht keine Speicherung alternativer Anfrageobjekte vor.

4.4.3 Aktualisierung des INQP-Teils

Ist eine Reoptimierung laufender Anfragen des INQP-Teils nicht auf Basis von bereits integrierten alternativen Anfrageplänen möglich, dann ist eine Aktualisierung der Laufzeitumgebung auf den Sensorknoten notwendig. Die Kosten bei dieser Aktualisierung werden durch zwei wesentliche Faktoren bestimmt: (i) die Menge der zu sendenden Daten und (ii) die Netzwerkgröße, welche die Anzahl der benötigten Nachrichten für eine vollständige Verteilung festlegt. Die Menge der zu sendenden Daten mem_{img} lässt sich nach Formel 4.4 bestimmen. Das Propagieren dieser Daten via OTA erfolgt üblicherweise auf Basis von Broadcast-Nachrichten. In einem Netzwerk der Höhe h mit einer mittleren Anzahl von f Kindknoten pro Sensorknoten ergeben sich somit für das Versenden und Einspielen auf den Sensorknoten Energiekosten in Höhe von

$$\mathsf{E}_{up}(mem_{img}) \quad = \quad (2 \cdot \mathsf{E}_{publish}(mem_{img}) + \mathsf{E}_{import}(mem_{img})) \cdot \sum_{i=1}^{h-1} f^i$$

wobei $\mathsf{E}_{publish}(mem_{img})$ die Kosten für das Senden bzw. Empfangen und $\mathsf{E}_{import}(mem_{img})$ die Kosten für das Einspielen der Laufzeitumgebung auf den Sensoren bezeichnet. Da die Laufzeitumgebung im Allgemeinen nicht mit einer Nachricht versendet werden kann, muss das Update in eine entsprechend hohe Anzahl einzelner Nachrichten zerlegt werden.

Sind die Kosten für das Aktualisieren der Laufzeitumgebung bekannt, so lässt sich ermitteln, ob und inwieweit das Einspielen einer gegebenen alternativen Laufzeitumgebung zu einer Laufzeitverlängerung des WSN führt. In einem ersten Schritt muss hierzu die verbleibende Lebenszeit Δt_{remain} des WSN bestimmt werden. Zu diesem Zweck müssen die im WSN verbliebenen Energiereserven (zum Beispiel durch Auslesen der Batteriekapazitäten) bestimmt werden. Da davon ausgegangen wird, dass die einzelnen Sensorknoten eine ähnliche Belastung erfahren, ist das Auslesen eines einzelnen Sensorknotens ausreichend. Anschließend kann unter Betrachtung des voraussichtlichen Energiebedarfs der gegenwärtig ausgeführten Konfiguration EXE_{old} bestimmt werden, wie hoch die verbleibende Lebenszeit ist.

Eine Aktualisierung der auf den Sensorknoten ausgeführten Laufzeitumgebung ist genau dann sinnvoll, wenn

$$\sum_{\bar{q} \in EXE_{old}} \mathsf{E}(\bar{q}, \Delta t_{remain}) > \sum_{\bar{q} \in EXE_{new}} \mathsf{E}(\bar{q}, \Delta t_{remain}) + \mathsf{E}_{update}(mem_{img})$$

EXE_{new} bezeichnet dabei die neue Laufzeitumgebung. Sowohl die verbleibende Energie als auch die notwendige Energie für die Ausführung der neuen Laufzeitumgebung kann nach Formel 4.2 bestimmt werden.

Es handelt sich hierbei um eine einfache Approximation, welche voraussetzt, dass alle Knoten im WSN der gleichen energetischen Belastung unterliegen. Dass diese Annahme nicht unrealistisch ist, zeigt sich am Beispiel der Entwicklung von effizienten Routingprotokollen. In [111] wird zum Beispiel eine Routingstrategie beschrieben, welche den verwendeten Routingpfad in Abhängigkeit der Auslastung kontinuierlich anpasst. Der in [170] beschriebene Ansatz adaptiert die Ausführungslokation von Berechnungen in Abhängigkeit der verbleibenden Kapazitäten auf den Sensorknoten.

Für das Problem einer effizienten Aktualisierung der Sensorknotensoftware existieren in der Literatur zahlreiche Ansätze [69, 118, 155, 217]. Diese betrachten sowohl den effizienten Transfer von Updates im WSN als auch Techniken, um Änderungen auf den Sensorknoten störungsfrei einzuspielen. Je nach Granularität der von der Plattform unterstützten austauschbaren Softwaremodule müssen die oben betrachteten Kostenformeln angepasst werden. So ist zum Beispiel eine Aktualisierung der Betriebssystemkomponente und der funktionalen Module nicht in jedem Fall erforderlich.

4.5 Multi-Sampling-Optimierung

Im Hinblick auf die stark beschränkten Ressourcen auf den Sensorknoten ist eine Wiederverwendung von Operatoren oder Teilanfragen auf der INQP-Komponente von besonderem Interesse. Im nachstehenden Abschnitt werden daher Möglichkeiten präsentiert, die Datenerzeugung auf den Sensorknoten über mehrere Anfragen hinweg zu optimieren. Prinzipiell kann dieses Problem als Teil einer Multi-Query-Optimierung angesehen werden. Daher wird zunächst in Abschnitt 4.5.1 ein Überblick über Techniken in diesem Bereich gegeben. Anschließend folgt in Abschnitt 4.5.2 eine Beschreibung von zwei Ansätzen zur anfrageübergreifenden Datenerfassung auf den Sensorknoten.

4.5.1 Multi-Query-Optimierung

Ziel der Multi-Query-Optimierung (MQO) ist das Ermitteln des optimalen (globalen) Anfrageplanes, welcher über allen gegenwärtig vom System ausgeführten Anfragen definiert ist. Das Problem der MQO wurde im Kontext lang laufender OLAP-Anfragen bereits ausführlich untersucht [230]. Die entsprechenden Verfahren lassen sich grundlegend kategorisieren:

- *lokale Anfrageoptimierung*: Auszuführende Anfragen werden in Teilanfragen zerlegt, welche vom System getrennt voneinander optimiert werden. Die optimalen lokalen Teilanfragen werden anschließend zu einer globalen Anfrage verbunden, wobei man versucht, so viele gemeinsame Teile in den Anfragen zu finden wie möglich [230].
- *globale Anfrageoptimierung*: Im Gegensatz zur lokalen Anfrageoptimierung, bei der das Optimierungsverfahren die Kosten lediglich für eine Teilanfrage minimiert, wird bei der globalen Optimierung die bestmögliche anfrageübergreifende

Lösung gesucht. Bei der globalen Anfrageoptimierung ist es möglich, dass suboptimale lokale Pläne miteinander verknüpft werden, welche anschließend in ihrer Kombination aber kosteneffizienter sind als die lokal optimierte Variante [190,230].

- *Erweiterung der globalen Anfrage*: Im Gegensatz zu den bisher genannten Verfahren geht dieser Ansatz von einem dynamischen Anfrageszenario aus, bei dem neue Anfragen getrennt betrachtet werden. Dabei sind alle aktuell laufenden Anfragen Teil der globalen Anfrage. Vom Nutzer neu definierte Anfragen werden auf Gemeinsamkeiten mit bereits laufenden Anfragen geprüft. Werden hierbei wiederverwendbare Teile identifiziert, welche zu einer Verbesserung der Gesamtperformance führen, dann werden die entsprechenden Anfragen angepasst. Das in [219] vorgestellte Verfahren vergleicht hierzu die hinzugekommene Anfrage mit jeder einzelnen bereits laufenden Anfrage und tauscht gegebenenfalls beide Anfragen gegen eine global kostengünstigere Anfrage aus.

In [116] wurde gezeigt, dass das MQO-Problem NP-hart ist und nur über heuristische Verfahren effizient gelöst werden kann. Viele in der Literatur zu findende (globale) Lösungsverfahren basieren daher auf einem Greedy-Ansatz. Werden wiederverwendbare Anfrageteile identifiziert, dann werden zuerst jene verwendet, welche den größten Kostengewinn versprechen. Mit den verbleibenden Anfrageteilen wird ebenso verfahren, bis alle weiteren Anfragen vollständig in die globale Anfrage integriert sind [218, 219, 230].

Neben der Kombination einfacher Filter-Operatoren steht bei der Datenstromverarbeitung insbesondere die Wiederverwendung von fensterbasierten Operatoren im Vordergrund. So versucht der in [90] vorgestellte Ansatz zum Beispiel Aggregat-Operationen zu synchronisieren. Ziel der in [214] präsentierten Arbeit ist die Wiederverwendung von fensterbasierten Verbund-Operatoren.

Im Bereich der WSNs gibt es ebenfalls erste Arbeiten zur MQO. Bei den Lösungen wird zwischen einer Optimierung auf der Basisstation (überwiegend für Langzeitanfragen verwendet) und einer In-Network-Optimierung unterschieden. Im Fall der In-Network-MQO findet die eigentliche Optimierung auf den Sensorknoten selbst statt, was zu entsprechenden Einschränkungen führt. So beschränken sich die Autoren in [202] zum Beispiel auf sehr kleine Anpassungen der Anfragen. Zumeist liegt der Schwerpunkt bei den Verfahren der In-Network-MQO auf einfachen Aggregatanfragen. Eine besondere Herausforderung sind hierbei die durch den Informationsaustausch zusätzlich anfallenden Kommunikationskosten [218].

4.5.2 Anfrageübergreifendes Sampling auf Sensorknoten

Im Bereich der OLAP-Systeme wurde gezeigt, dass ein suboptimal lesender Datenzugriff bei gleichzeitiger Wiederverwendung in der globalen Anfrage kosteneffizienter sein kann als das Verwenden des optimalen Zugriffs für jede einzelne Anfrage [230].

Ein ähnliches Problem ergibt sich beim Sampling auf den Sensorknoten. Dieses Erzeugen von Daten zählt neben dem Versenden zu den energieintensivsten Operationen.

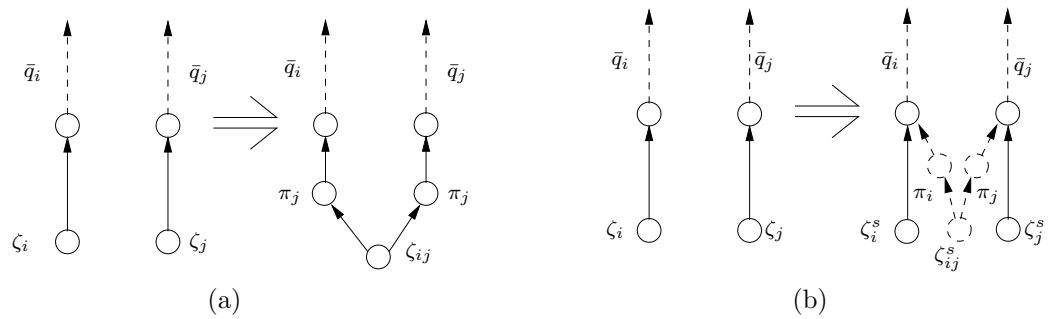


Abbildung 4.10: Zusammenfassen zweier Sampling-Operatoren: (a) einfache Zusammenfassung, (b) zyklische Zusatzmessungen

Jedes Vermeiden von Messungen durch die physischen Sensoren vermindert somit den Energieverbrauch enorm. Ein einfacher Ansatz, unnötige Messungen zu vermeiden, ist die Wiederverwendung von erfassten Werten in anderen Anfragen bzw. Anfrageteilen auf den Sensorknoten, welche gleiche Daten produzieren. Die Quellen ζ_i und ζ_j werden dabei zu einer gemeinsamen Quelle ζ_{ij} zusammengefasst (siehe Abbildung 4.10(a)). Der in *AnduIN* verwendete *subscribe-unsubscribe*-Mechanismus ist für die Wiederverwendung von bereits gemessenen Werten in weiteren Anfrageteilen besonders geeignet. Jeder Operator kann sich hierzu bei bereits existierenden Quellen einfach zusätzlich registrieren.

Die resultierende gemeinsame Quelle ζ_{ij} muss alle Attribute erfassen, welche zuvor von den einzelnen Quellen gesampelt wurden, d.h. das Schema von ζ_{ij} entspricht dem Schema von $\zeta_i \cup \zeta_j$. Um die geforderten Schemata für die folgenden Operatoren wiederherzustellen, müssen nach dem gemeinsamen Sampling-Operator zusätzliche Projektionen eingeführt werden. Die zusätzlichen Projektionen wiederum sind mit weiteren Kosten (Energie und Speicher) verbunden. Diese fallen dabei im Allgemeinen deutlich geringer aus als die gesparten Kosten für das mehrfache Sampling.

Das Zusammenführen unterschiedlicher Sampling-Operatoren zu einer gemeinsamen Quelle ist nicht nur bei sich überlappenden Schemata sinnvoll. Sind physische Sensoren hardwaretechnisch nicht vollständig voneinander getrennt, so kann auch das zeitgleiche Erfassen von miteinander verbundenen physischen Sensoren effizienter sein als das getrennte Messen. Beispielsweise ist das gemeinsame Erfassen von Luftfeuchtigkeit und Temperatur bei den für diese Arbeit eingesetzten Sensoren mit $4738.8\mu J$ verbunden. Für eine getrennte Erfassung sind $3753.4\mu J + 1655.3\mu J = 5408.7\mu J$ erforderlich (siehe Tabelle 6.1). Werden also beide Werte angefordert, dann ist ein gemeinsames Messen mit geringeren Kosten die bessere Alternative.

Ein Problem beim Kombinieren zweier oder mehrerer Sampling-Operatoren ergibt sich, falls die Operatoren unterschiedliche Samplingintervalle verwenden. Für den Umgang mit diesem Problem sollen im Weiteren zwei Ansätze präsentiert werden: (i) Adaption der Samplingperiode und (ii) Synchronisation durch zusätzliche Quellen.

4.5.2.1 Abgeschwächte Sampling-Perioden

Oftmals ist es für den Anwender ausreichend, wenn das Sampling innerhalb eines definierten Zeitraumes stattfindet. Beim Ansatz der abgeschwächten Sampling-Perioden definiert er hierzu neben den eigentlichen Sampling-Perioden zusätzlich noch eine maximale Abweichung. Diese definiert, inwieweit die Sampling-Periode von dem definierten Wert abweichen darf. Dieser Wert erlaubt dem System anschließend, im Falle einer Ressourcen-Einsparung die automatische Anpassung der Sampling-Periode durch Kombination von Sampling-Operatoren.

Das System versucht die für die einzelnen Sampling-Operatoren spezifizierten Sampling-Intervalle soweit anzupassen, dass die kombinierte Sampling-Periode allen Quell-Operatoren genügt, wobei aber die vom Nutzer definierten Schwellen nicht überschritten werden dürfen. In [149] wurde ein ähnlicher Ansatz erwähnt, welcher allerdings nicht konkret ausgeführt wurde. Im Folgenden soll eine Lösung präsentiert werden, welche auf der Operatorsemantik von *AnduIN* und dem damit verbundenen Kostenmodell basiert.

Kostenersparnis Zunächst soll bestimmt werden, wie groß die Kostenersparnis durch das Zusammenfassen zweier Sampling-Operatoren ζ_i und ζ_j zu einem Operator ζ_{ij} ist. Aufgrund der angepassten Sampling-Periode ändern sich u.U. die Ausführungshäufigkeiten aller von den Sampling-Operatoren abhängigen Operatoren. Die Kostenersparnis durch die Kombination der Sampling-Operatoren lässt sich entsprechend wie folgt ermitteln:

$$p(\bar{q}', \bar{q}'', \zeta_i, \zeta_j) = \begin{cases} \mathsf{E}(\bar{q}') + \mathsf{E}(\bar{q}'') - \mathsf{E}(\bar{q}'_{\zeta_i \to \zeta_{ij} \wedge \pi_i}) - \mathsf{E}(\bar{q}''_{\zeta_j \to \zeta_{ij} \wedge \pi_j}) & : \bar{q}' \neq \bar{q}'' \\ \mathsf{E}(\bar{q}') - \mathsf{E}(\bar{q}'_{\zeta_i \wedge \zeta_j \to \zeta_{ij} \wedge \pi_i}) & : \bar{q}' = \bar{q}'' \end{cases} \tag{4.5}$$

$\mathsf{E}(\bar{q})$ bezeichnet den Energieverbrauch der Anfrage $\bar{q}$ im Zeitintervall Δt (nach Gleichung 4.2). Alle Berechnungen erfolgen über dem gleichen Zeitintervall. Aus Gründen der Übersichtlichkeit wurde Δt in obiger Gleichung weggelassen. $\bar{q}'$ bzw. $\bar{q}''$ bezeichnen die Anfragen, welche ζ_i bzw. ζ_j als Quellen haben. $\bar{q}'$ und $\bar{q}''$ müssen nicht zwingend verschiedene Anfragen sein. So ist es auch möglich, das ζ_i und ζ_j Sampling-Operatoren der gleichen Anfrage sind und diese durch eine gemeinsame Quelle ersetzt werden. $\bar{q}'_{\zeta_j \to \zeta_{ij} \wedge \pi_j}$ ist die Anfrage, welche aus $\bar{q}'$ entsteht, wenn der Operator ζ_j durch den Operator ζ_{ij} und die Projektion π_j ersetzt wird.

Zusammenfügen von Operatoren Die Sampling-Periode des neuen Operators ζ_{ij} lässt sich wie folgt bestimmen: Es seien für die beiden Operatoren ζ_i und ζ_j mit den beiden Sampling-Perioden $s(\zeta_i)$ bzw. $s(\zeta_j)$ vom Anwender definiert. Zusätzlich wurden für jeden Operator mit $rx(\zeta_i)$ und $rx(\zeta_j)$ die maximal erlaubten Abweichungen der Sampling-Periode angegeben. Damit lässt sich für jeden Operator ein Intervall ermitteln, innerhalb dessen die Messungen stattfinden müssen:

$$I(\zeta_i) = \big[s(\zeta_i) - rx(\zeta_i), s(\zeta_i) + rx(\zeta_i)\big]$$

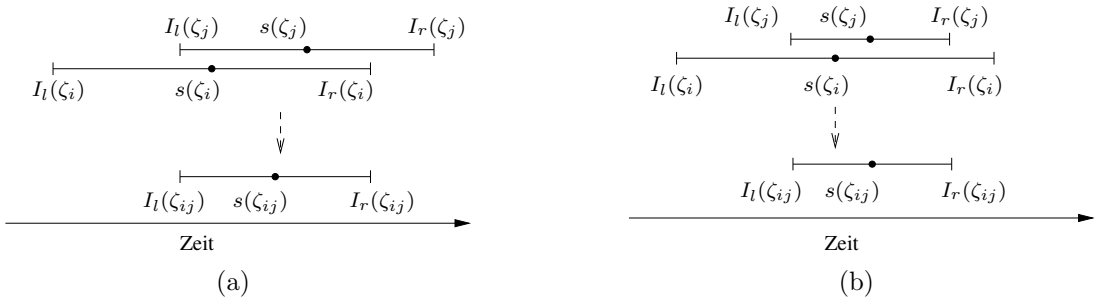


Abbildung 4.11: Abgeschwächte Sampling-Perioden: (a) überlappende Sampling-Intervalle, (b) überdeckende Sampling-Intervalle

Basierend auf den Intervallen für ζ_i und ζ_j kann somit die Sampling-Periode und die maximal erlaubte Abweichung für ζ_{ij} bestimmt werden. Die Abbildungen 4.11(a) und 4.11(b) zeigen die beiden wichtigsten Fälle, die beim Kombinieren von ζ_i und ζ_j auftreten können.

Formal lässt sich die neue Sampling-Periode für Operator ζ_{ij} wie folgt bestimmen (o.B.d.A. sei $s(\zeta_i) \leq s(\zeta_j)$):

$$s(\zeta_{ij}) = \begin{cases} \frac{I_r(\zeta_i)+I_l(\zeta_j)}{2} & , I_l(\zeta_i) < I_l(\zeta_j) \wedge I_r(\zeta_i) < I_r(\zeta_j) \\ s(\zeta_i) & , I_l(\zeta_j) < I_l(\zeta_i) \wedge I_r(\zeta_i) < I_r(\zeta_j) \\ s(\zeta_j) & , I_l(\zeta_i) \leq I_l(\zeta_j) \wedge I_r(\zeta_j) \leq I_r(\zeta_i) \\ \emptyset & sonst \end{cases} \tag{4.6}$$

I_l bezeichnet dabei die linke und I_r die rechte Intervallgrenze der Relaxations-Intervalle. Die maximal mögliche Abweichung von der Sampling-Periode für ζ_{ij} kann analog bestimmt werden:

$$rx(\zeta_{ij}) = \begin{cases} s(\zeta_{ij}) - I_l(\zeta_j) & , I_l(\zeta_i) < I_l(\zeta_j) \wedge I_r(\zeta_i) < I_r(\zeta_j) \\ rx(\zeta_i) & , I_l(\zeta_j) < I_l(\zeta_i) \wedge I_r(\zeta_i) < I_r(\zeta_j) \\ rx(\zeta_j) & , I_l(\zeta_i) \leq I_l(\zeta_j) \wedge I_r(\zeta_j) \leq I_r(\zeta_i) \\ \emptyset & sonst \end{cases} \tag{4.7}$$

Ein einfaches Beispiel soll das Zusammenführen zweier Sampling-Operatoren verdeutlichen.

Beispiel 4.6 Gegeben seien zwei Quell-Operatoren ζ_1 und ζ_2 mit $s(\zeta_1) = 5$, $rx(\zeta_1) = 1$, ζ_2 $s(\zeta_2) = 7$ und $rx(\zeta_2) = 1.4$. Es wird angenommen, es gelte $p(\zeta_1, \zeta_2) > 0$. Dann können die beiden Operatoren durch den Operator ζ_{12} mit $s(\zeta_{12}) = (6 + 5.6)/2 = 5.8$ und $rx(\zeta_{12}) = 5.8 - 5.6 = 0.2$ ersetzt werden. ■

Aufgrund der möglicherweise großen Anzahl an Kandidaten für das Zusammenführen ist die Suche nach der optimalen Lösung nicht immer effizient möglich. Es müssen nicht nur die Sampling-Operatoren der Originalanfragen untereinander auf mögliche Überschneidungen geprüft werden, sondern auch bereits kombinierte Operatoren. Algorithmus 4.1 zeigt einen Greedy-Ansatz, welcher die Menge der Sampling-Operatoren auf

Input : $\vec{q}', \vec{q}''$ // *Original-Anfragen*
Output : $\vec{q}', \vec{q}''$ // *Anfragen mit kombinierten Sampling Operatoren*

```
repeat
    p = 0;
    ζ_best, S̄_ζ^drop = ∅;
    /* suche Sampling-Operator, welcher Kostenersparnis maximiert */
    for ζ' ∈ q⃗' do
        for ζ'' ∈ q⃗'' do
            // erzeuge Sampling- und Relaxations-Intervall für ζ'_ij
            ζ^new = mergeSamplingOps(ζ', ζ''); // nach Gleichung 4.6 und 4.7
            p^new = computeProfit(q⃗', q⃗'', ζ', ζ'', ζ^new); // nach Gleichung 4.5
            if p^new > p then
                p = p^new;
                ζ_best = ζ^new;
                S̄_ζ^drop = {ζ', ζ''};
    /* tausche Operatoren aus */
    if ζ_best ! = ∅ then
        // tauscht die Operatoren S̄_ζ^drop in den Anfragen q⃗', q⃗'' durch
        // ζ_best und eventuell zusätzliche Projektionen aus
        replaceOldOps(q⃗', q⃗'', S̄_ζ^drop, ζ_best) ;
until ζ_best ! = ∅ ;
return S̄_ζ;
```

Algorithmus 4.1 : Kombinieren von Sampling-Operatoren durch Abschwächen der Sampling-Perioden

mögliche Paare zu vereinender Operatoren hin prüft und aus diesen dasjenige Operatorpaar auswählt, welches die voraussichtliche Kostenersparniss maximiert. Die vereinten Operatoren werden aus den Anfragen entfernt und durch den kombinierten Operator ersetzt. Das Verfahren wird solange auf die verbleibenden Sampling-Operatoren angewendet, bis sich keine zwei Operatoren mehr finden lassen, welche durch einen kosteneffizienteren Operator ersetzt werden können.

Der wesentliche Nachteil dieses Ansatzes ist die Verwendung von veränderten Sampling-Perioden. Dies ist zumeist mit einem Informationsverlust verbunden, da prinzipiell weniger Werte pro Zeiteinheit erfasst werden können. Liegen die Intervalle zweier Kandidaten zu weit auseinander, dann lassen sich erst gar keine entsprechenden Überlappungen zwischen diesen finden. Ein Verknüpfen von zwei Sampling-Operatoren ist nicht möglich und das Sparpotential kann nicht ausgeschöpft werden. Der nachstehende Ansatz versucht diese Probleme auf Kosten einer möglicherweise geringeren Kostenersparnis bzw. eines erhöhten Speicherbedarfs zu lösen.

4.5.2.2 Synchronisation durch zusätzliche Sampling-Operatoren

Beim Ansatz der Synchronisation durch extra Quellen wird zusätzlich zu zwei vorhandenen Sampling-Operatoren ζ_i und ζ_j, für die $p^+(\zeta_i, \zeta_j) > 0$ gilt, ein *zusätzlicher*

```
   Input : S̄_ζ = {ζ_1, ζ_2, ...}, maxmem // Menge aller Sampling-Operatoren
   Output : S̄_ζ = {ζ'_1, ζ'_2, ...} // Menge der optimierten Sampling-Operatoren
 1 repeat
 2     p = 0; S̄_ζ^adapt = ζ^best = ∅;
 3     /* suche Sampling Operator, welcher Kostenersparnis maximiert */
 4     for ζ' ∈ S̄_ζ do
 5         for ζ'' ∈ S̄_ζ/ζ' do
 6             ζ^new = deriveNewSampl(ζ', ζ''); // Sampling-Periode ergibt sich aus kgV
 7             p' = computeProfit(ζ', ζ'', ζ^new); // nach Gleichung 4.8
 8             if (p' > p) ∧ (maxmem − mem_img(ζ^new) > 0) then
 9                 p = p';
10                 ζ^best = ζ^new;
11                 S̄_ζ^adapt = {ζ', ζ''};
12     /* füge neuen Sampling Operator hinzu */
13     /* entferne eventuell nicht mehr notwendige Sampling Operatoren */
14     if ζ^best ! = ∅ then
15         maxmem = maxmem − mem_img(ζ'_ij);
16         S̄_ζ = S̄_ζ ∪ ζ_best;
17         adaptSamplFreq(S̄_ζ, ζ_best); // passt Sampling an
18         prune(S̄_ζ); // entfernt eventuell unnötige Operatoren
19 until ζ^best ! = ∅ ∧ maxmem > 0 ;
20 return S̄_ζ;
```

Algorithmus 4.2 : Kombinieren von Sampling-Operatoren durch Einfügen zusätzlicher Operatoren

Operator ζ_{ij} eingeführt (siehe Abbildung 4.10(b)). Dieser misst genau dann anstelle der Original-Operatoren ζ_i und ζ_j, wenn diese zum gleichen Zeitpunkt aktiviert werden würden. Der gemeinsame Messzeitpunkt lässt sich einfach durch das kleinste gemeinsame Vielfache der einzelnen Sampling-Zeitpunkte $kgV(s(\zeta_i), s(\zeta_j))$ ermitteln (unter der Annahme, dass der Start beider Operatoren synchron erfolgt).

Kostenersparnis Die Kostenersparnis durch das Einfügen eines zusätzlichen Sampling-Operators lässt sich wie folgt bestimmen :

$$p^+(\zeta_i, \zeta_j) \quad = \quad \frac{s(\zeta_i) \cdot \mathsf{E}(\zeta_i) + s(\zeta_j) \cdot \mathsf{E}(\zeta_j)}{kgV(s(\zeta_i), s(\zeta_j))} - \Big(\mathsf{E}(\zeta_{ij}) + \mathsf{E}(\pi_i) + \mathsf{E}(\pi_j)\Big) \qquad (4.8)$$

Die Berechnung der Energiekosten erfolgt für jeden Operator über dem selben Zeitintervall Δt. Aus Gründen der Übersichtlichkeit wurde das Argument Δt in obiger Formel wieder weggelassen.

Aufgrund der stetig wachsenden Menge an möglichen Sample-Operatoren pro Iterationsschritt (jeder der existierenden Sampling-Operatoren kann mit jedem der zusätzlichen Operatoren kombiniert werden) kommt auch bei diesem Ansatz ein Greedy-Verfahren zum Einsatz (siehe Algorithmus 4.2). Hierbei sind allerdings zwei Dinge zu beachten:

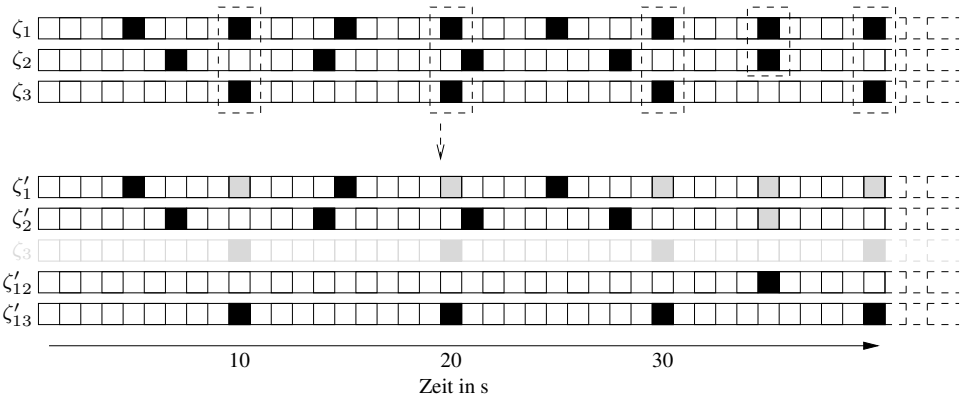


Abbildung 4.12: Beispiel für das Einfügen zusätzlicher Sampling-Operatoren

- Das Einfügen zusätzlicher Operatoren ist mit einem zusätzlichen Bedarf an Speicher verbunden. Um dem Anfrageplan nicht alle sinnvollen zusätzlichen Sampling-Operatoren hinzuzufügen und somit den Speicherbedarf entsprechend zu erhöhen, kann eine Obergrenze *maxmem* für den maximal zusätzlich zu belegenden Speicher spezifiziert werden. Sofern nichts anderes definiert wurde, entspricht *maxmem* dem noch maximal in der Laufzeitumgebung zur Verfügung stehenden Speicher.

- Die Kostenersparnis durch Einfügen zusätzlicher Sampling-Operatoren hängt nicht mehr nur von der Art des eingefügten Operators ab. Zusätzlich beeinflusst die Häufigkeit, mit der er zum Einsatz kommt, den Energiegewinn. So kann zum Beispiel die Kostenersparnis durch den Einsatz eines einzelnen zusätzlichen Operators geringer sein als die durch das Einfügen eines anderen zusätzlichen Operators. Wird dieser aber öfter ausgeführt als letzterer, so kann die Gesamtersparnis durchaus größer sein.

Neben der Sampling-Periode müssen bei diesem Ansatz für jeden Operator zusätzlich die Zeitpunkte gespeichert werden, bei denen das Messen ausgelassen wird. Überdecken die auszulassenden Sampling-Zeitpunkte alle originalen Sampling-Zeitpunkte eines Operators, so dann kann dieser aus der Menge der Sampling-Operatoren entfernt werden (*pruning*). Ein Beispiel soll den Ansatz der Kostenersparnis durch Hinzufügen zusätzlicher Operatoren verdeutlichen:

Beispiel 4.7 Gegeben sind drei Sampling-Operatoren ζ_1, ζ_2 und ζ_3 mit $s(\zeta_1) = 5$, $s(\zeta_2) = 7$ und $s(\zeta_3) = 10$. Aus diesen lassen sich die zusätzlichen Operatoren ζ_{12}, ζ_{13} und ζ_{23} mit $s(\zeta_{12}) = 35$, $s(\zeta_{13}) = 10$ und $s(\zeta_{23}) = 70$ direkt ableiten. Weiterhin wird angenommen, dass maximal ein zusätzlicher Operator in die Laufzeitumgebung integriert werden kann und dass durch ζ_{13} die größte und durch ζ_{23} die geringste Kostenersparnis erzielt wird.

Zunächst wird ζ_{13} in die Laufzeitumgebung integriert. Daraufhin kann aus ζ_1 jedes zweite und aus ζ_3 jedes Sampling entfernt werden. Da der Operator ζ_3 nach dem Entfernen aller Samples nicht mehr notwendig ist, kann er komplett aus der Laufzeitumgebung

entfernt werden. Dadurch wird Speicher frei, welcher das Hinzufügen eines weiteren kombinierten Sampling-Operators erlaubt. Aufgrund der größeren Kostenersparnis wird ζ_{12} in die Laufzeitumgebung integriert. Daraufhin wird aus ζ_1 jedes siebente Sampling entfernt und aus ζ_2 jedes fünfte. Die resultierenden Samples sind in Abbildung 4.12 dargestellt. Grau dargestellte Samples bezeichnen nach dem Kombinieren nicht mehr notwendige Messungen. ■

Die beiden oben beschriebenen Verfahren können auch in Kombination verwendet werden. In einem ersten Durchlauf werden dabei Sampling-Operatoren in den definierten Schranken durch zusammenfassende Operatoren ersetzt. In einem zweiten Durchlauf wird versucht, die resultierenden Operatoren des ersten Durchlaufes, falls möglich, mittels zusätzlicher Operatoren zu synchronisieren.

Der Einsatz zusätzlicher (synthetischer) Quellen ähnelt dabei dem Ansatz von Xiang und Lim in [220]. Die Autoren beziehen sich dabei jedoch auf die Einführung einer synthetischen Anfrage, welche der Multi-Query-Optimierung dient. Eine Betrachtung auf Ebene der Sampling-Operatoren wurde von ihnen nicht durchgeführt.

4.6 Zusammenfassung

Das zurückliegende Kapitel widmete sich der Anfrageverarbeitung in *AnduIN*. Hierzu wurde zunächst das bekannte Operatorgraphen-Modell vorgestellt und um die Verarbeitung komplexer Operatoren, welche teilweise im WSN verarbeitet werden können, erweitert. Anschließend wurde ausführlich das in *AnduIN* integrierte zweidimensionale Kostenmodell beschrieben. Im Weiteren erfolgte eine Beschreibung der initialen Anfrageoptimierung, gefolgt von einem Überblick über mögliche Reoptimierungsansätze. In beiden Fällen wurde auf auftretende Probleme im Zusammenhang mit der partiell verteilten Arbeitsweise von *AnduIN* eingegangen, entsprechende Lösungsansätze wurden präsentiert. Abschließend wurde noch das Problem der anfrageübergreifenden Optimierung diskutiert. Hierzu wurden ebenfalls verschiedene existierende Ansätze präsentiert und zusätzliches Optimierungspotential im Bereich der Sensornetzwerke aufgezeigt. Das nächste Kapitel widmet sich konkreten, in *AnduIN* integrierten Operatoren und deren physischer Umsetzung.

Kapitel 5

Operatoren

Nachdem im letzten Kapitel auf die Anfrageverarbeitung in *AnduIN* eingegangen wurde, sollen im folgenden Abschnitt die vom System zur Verfügung gestellten Operatoren im Detail betrachtet werden. Hierbei soll insbesondere auf neu entwickelte Verfahren eingegangen werden.

Das in der Arbeit entwickelte System stellt alle bekannten primitiven Operatoren wie Filter, Projektion, Verbund, Aggregationen (zum Beispiel min, max, avg, sum, ...), Gruppierung und Having zur Verfügung. Abgesehen von der Gruppierung und von der Having-Klausel stehen dabei alle genannten Operatoren sowohl auf der DSMS- als auch auf der INQP-Komponente zur Verfügung. Für den Verbund existieren dabei jeweils sowohl eine Nested-Loop-Join-Implementierung als auch eine Hash-Join-Implementierung.

Zusätzlich zu diesen wurden weitere verschiedene Operatoren integriert, die zum überwiegenden Teil in Form von Synopsen-Operatoren vorliegen. Tabelle 5.1 liefert einen Überblick über die zusätzlich in *AnduIN* integrierten Operatoren. Die Tabelle zeigt zudem, dass ein Teil der Operatoren vollständig bzw. teilweise für die In-Network-Komponente entwickelt wurde.

Logischer Operator	Physische Operatoren			Synopsen Operator
	DSMS	INQP	DSMS/INQP	
räumliche Operatoren	√	√	-	-
adaptive Burst Erkennung [127]	√	√	-	√
Clustering (CluStream [7])	√	-	-	√
exponentielle Glättung	√	-	-	√
FPP Stream [128]	√	-	√	√
Lossy Counting	√	-	-	√
Fehlwerte Erkennung	√	-	-	√
Ausreißer Erkennung	√	√	-	√

Tabelle 5.1: Erweiterte Operatorbibliothek von *AnduIN*

Im weiteren Verlauf dieses Kapitels werden Operatoren beschrieben, welche sich den nachstehenden Schwerpunkten zuordnen lassen:

- *räumliche Operatoren*: Anfragen mit räumlichem Bezug sind gerade im Bereich der räumlich verteilten WSNs sinnvoll, stellen aber zugleich eine große Herausforderung dar. Zum einen bietet eine frühzeitige Filterung von Daten auf Basis räumlicher Informationen ein großes Energie-Einsparpotential. Zum anderen besteht aber auch die Schwierigkeit, geeignete Datenstrukturen zur effizienten räumlichen Filterung sowohl im DSMS-Teil als auch im INQP-Teil zu entwickeln. In Abschnitt 5.1 soll daher untersucht werden, wie sich räumliche Anfragen in *AnduIN* integrieren lassen und effizient verarbeitet werden können.

- *adaptive Burst-Erkennung*: Ein bei der Verarbeitung von Sensordaten häufig auftretendes Problem ist die geringe Datenqualität. Aufgrund von Fehlern in der Sensorik oder bei der Übertragung von Daten kann es zu Datenverlust oder Datenfehlern kommen. Entsprechende Verfahren zur Datenbereinigung sind daher ein Schwerpunkt bei der Sensordatenverarbeitung. Im Rahmen dieser Problemklasse soll in Abschnitt 5.2 ein Verfahren zur *adaptiven Burst-Erkennung* beschrieben werden. Ziel des Verfahrens ist die Erkennung von abnormalem Verhalten über nicht-stationären Daten.

- *quantitatives Frequent Pattern Mining*: Die In-Network-Verarbeitung komplexer Verfahren ist nicht immer vollständig möglich oder sinnvoll. Stellvertretend für die Klasse der teilweise im WSN verarbeiteten Algorithmen soll in Abschnitt 5.3 ein Verfahren für das Finden häufiger Muster präsentiert werden. Das Problem des Frequent Pattern Mining gehört zu den klassischen Data-Mining-Verfahren. Im Gegensatz zu existierenden Verfahren, welche auf die Analyse kategorischer Attribute beschränkt sind, soll der hier vorgestellte Ansatz häufige Muster über quantitativen Attributen identifizieren. Die auf Sensorknoten bestimmten Messwerte sind typischerweise dieser Art.

Die einzelnen Abschnitte zu den jeweiligen Operatoren sind wie folgt gegliedert: Zunächst wird die mit den vorgestellten Operatoren verbundene Problemstellung kurz motiviert. Anschließend werden notwendige Techniken und Begriffe erläutert. Auf dieser Grundlage werden die im Rahmen dieser Arbeit entwickelten Algorithmen und Datenstrukturen präsentiert und deren Integration in *AnduIN* beschrieben. Hierbei wird insbesondere auf die Problematik der In-Network-Verarbeitung der vorgestellten Verfahren eingegangen. In einem abschließenden Abschnitt wird jeweils ein kurzer Überblick über Vorarbeiten im Bereich der Aufgabenstellung präsentiert.

5.1 Anfragen mit räumlichem Bezug

Die zivile Nutzung von GPS (*Global Positioning System*) führte zu einer breiten Nutzung von Daten mit räumlichem Bezug. So sind moderne Fahrzeuge heute oftmals be-

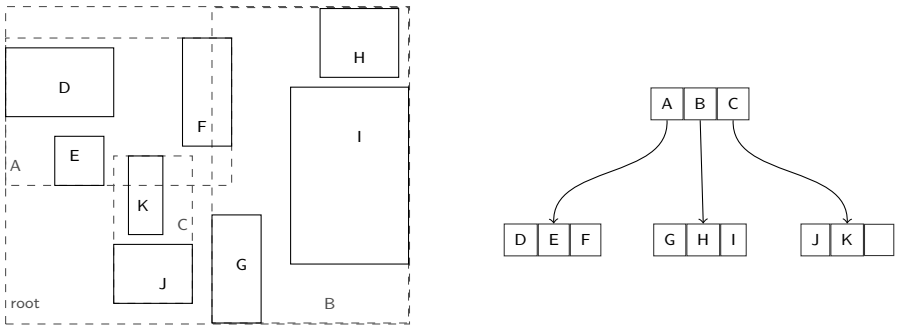
reits mit entsprechenden Navigationssystemen ausgestattet, welche dem Fahrzeugführer eine einfache, zielgerichtete Routenplanung und -verfolgung ermöglichen. Räumliche Daten sind aber nicht erst seit der Einführung von GPS im kommerziellen Bereich von Interesse. So gab es bereits vorher die Möglichkeit, den Standort von Objekten mehr oder weniger genau zu bestimmen (zum Beispiel durch Triangulation über Mobilfunkzellen oder manuelles Vermessen). Bereits in den frühen 60er Jahren des letzten Jahrhunderts wurde mit dem „Canada Geographic Information System“ ein erstes Geoinformationssystem (GIS) zur Überwachung der kanadischen Landressourcen entwickelt. Mittels dieses Systems wurden dabei Informationen über die Kapazitäten des Landes (Wälder, Seen, Flora und Fauna sowie Bodenschätze) gesammelt und analysiert.

Die Verwaltung solcher Daten in Datenbanksystemen versprach eine Vielzahl neuer und interessanter Analysemöglichkeiten. Sind zum Beispiel Objekte in einer Warenwirtschaft zusätzlich mit Geoinformationen angereichert, so lassen sich hieraus u.U. verfahrensoptimierende Aussagen über Unternehmensprozesse treffen. Mögliche Anfrage-Szenarien sind zum Beispiel „Wie viele Bauteile vom Typ X finden sich in unseren Werken im Umkreis von Y km?“ oder „Wie weit ist Fahrzeug X noch von Werk Y entfernt?“ Basierend auf diesen Analysen können Wartezeiten bestimmt oder weitere Fertigungsprozesse gestartet werden. Da die von den DBMS zur Verfügung gestellten Datentypen und Operatoren für eine exakte Geoanalyse oftmals nicht ausreichend waren, mussten entsprechende Erweiterungen geschaffen werden. Heute verfügen die meisten kommerziellen DBMS sowie eine Vielzahl von Open-Source-DBMS über entsprechende Erweiterungen.

Im weiteren Verlauf dieses Abschnittes soll daher untersucht werden, inwieweit eine Analyse von Daten mit räumlichem Bezug in *AnduIN* effektiv umsetzbar ist. Zunächst werden hierzu aus dem Bereich der DBMS existierende Lösungen vorgestellt und hinsichtlich ihrer Eignung für eine Integration geprüft. Anschließend wird untersucht, welche Operatoren und räumlichen Objekte im Fall der Datenstromverarbeitung sinnvoll sind und wie diese in das vorgestellte System integriert werden können. Des Weiteren soll auf Besonderheiten im Zusammenhang mit der Verarbeitung im INQP-Teil von *AnduIN* eingegangen werden. Abschließend wird ein Überblick über verwandte Arbeiten zum Thema gegeben.

5.1.1 Vorbetrachtungen

Betrachtet man Geoinformationssysteme, dann stellt sich zunächst die Frage nach der Wahl des Bezugssystems für eine einheitliche Handhabung der räumlichen Informationen. Im folgenden Abschnitt sollen hierzu kurz die bekanntesten Systeme vorgestellt werden. Der effiziente Umgang bei der Auswertung von Geoinformationen wird bereits seit einigen Jahren untersucht. Zu den grundlegenden Ansätzen zählt hierbei die Verwendung effizienter Indexstrukturen, welche im zweiten Teil dieses Abschnittes betrachtet werden sollen.



(a) (b)

Abbildung 5.1: Beispiel R-Baum: (a) Regionenaufteilung und (b) Datenstruktur

5.1.1.1 Koordinatensysteme

Um die eindeutige Position eines Objektes im Raum bestimmen zu können, ist ein entsprechendes Referenzsystem vonnöten. Das in der Mathematik am weitesten verbreitete Koordinatensystem ist das kartesische Koordinatensystem, welches häufig zur Beschreibung geometrischer Sachverhalte eingesetzt wird. Für die Betrachtung kugelförmiger Oberflächen wie zum Beispiel der Erde ist das kartesische Koordinatensystem ungeeignet. Hier wird zumeist auf krummlinige orthogonale sphärische Koordinatensysteme wie zum Beispiel das geographische Koordinatensystem zurückgegriffen. Für die Beschreibung eines Punktes auf der Kugeloberfläche sind zwei Winkel bezüglich des Koordinatenursprungs ausreichend. Im geographischen Koordinatensystem erfolgt die Beschreibung eines Punktes anhand seines Breiten-(*Latitude*) und seines Längengrades (*Longitude*). Beide werden in Relation zu entsprechenden Bezugspunkten (Null-Meridian bzw. Äquator) angegeben.

Ist neben der Beschreibung der Breiten- und Längenangabe noch eine Höhenangabe von Interesse, dann wird zumeist das geozentrische Koordinatensystem eingesetzt. Die Angaben beziehen sich dabei jeweils auf den Koordinatenursprung des Systems [197]. Die drei Achsen sind dabei jeweils senkrecht zueinander ausgerichtet. Das für die Positionsbestimmung entwickelte US-Satelitensystem NAVSTAR GPS basiert auf einem geozentrischen Koordinatensystem.

5.1.1.2 Indexstrukturen

Eine typische Klasse räumlicher Anfragen ist der Verbund räumlicher Objekte (*Spatial Join*). Formal handelt es sich beim Spatial Join um den Verbund zweier Mengen von Geo-Objekten G_1 und G_2 über einem raumbezogenen Prädikat θ. Die Ergebnismenge des Verbundes umfasst dabei alle Paare $(g_1, g_2) \in G_1 \times G_2$, für die $\theta(g_1, g_2)$ gilt. Häufig verwendete Prädikate θ sind *intersect*, *contains* und *distance*. Ein typisches Beispiel ist der Test räumlicher Objekte (zum Beispiel Sensorknoten) auf deren Existenz in einer

bestimmten Region (zum Beispiel einem Waldgebiet). Der Verbund findet dabei über geografischen Koordinaten statt.

Prinzipiell handelt es sich beim Spatial Join um eine sehr kostenintensive Operation. Die Kosten hängen dabei sowohl von der Größe der zu vergleichenden Datenmenge als auch von der Komplexität der einzelnen räumlichen Objekte ab. Um diese Kosten zu minimieren, wird häufig auf einen zweistufigen Prüfprozess zurückgegriffen. Hierbei wird zunächst während eines *filter*-Schrittes eine Menge von Kandidaten-Objekten bestimmt, welche dem verwendeten Prädikat eventuell genügt. Alle anderen Objekte werden verworfen. Anschließend wird die Kandidatenmenge während eines *refine*-Schrittes mit den exakten Objekten verglichen, d.h. die Kandidaten bezüglich des entsprechenden Prädikates geprüft. Der *filter*-Schritt basiert im Allgemeinen auf der Ausnutzung von Indexstrukturen. Hierdurch wird die Kandidaten-Menge zumeist stark reduziert, was wiederum die Kosten für den anschließenden *refine*-Schritt drastisch senken kann.

Seit den ersten Ansätzen zur Integration von raumbezogenen Objekten in Datenbanken wurden zahlreiche Indexstrukturen geschaffen. Für eine entsprechende Erweiterung zur Verarbeitung räumlicher Anfragen in *AnduIN* ist aus diesen eine geeignete auszuwählen. Daher wird im Folgenden ein kurzer Überblick über die wichtigsten Kandidaten gegeben. Einen ausführlichen Überblick liefert [79].

R-Baum Bei dieser Indexstruktur [96] wird der Raum in nichtdisjunkte rechteckige Teilregionen aufgeteilt. Die einzelnen Knoten enthalten dabei rechteckige Regionen (*minimum bounding box* – MBB), welche alle Regionen im Teilbaum unterhalb des Knotens umfassen. Die räumlichen Objekte finden sich in den Blattknoten. Für eine schnelle Suche liegen die einzelnen Objekte eines Knotens sortiert vor. Der Baum ist so organisiert, dass geografisch benachbarte Objekte auch im R-Baum nahe beieinander abgelegt werden. In Abbildung 5.1 sind exemplarisch Objekte (bzw. deren MBB) im $\mathbb{R}^2$ abgebildet.

Zu den kostenintensivsten Operationen im R-Baum zählt das Einfügen. Als problematisch stellen sich hierbei die Erweiterung von Regionen zur Aufnahme der neuen Region und das Aufspalten von Regionen im Fall eines Knotenüberlaufs heraus. Aufgrund der geforderten Ausgeglichenheit der Indexstruktur und der Notwendigkeit, Änderungen an Regionen von den Blättern zur Wurzel zu propagieren, kann es eventuell notwendig werden, Teilbäume bzw. den kompletten Baum beim Einfügen eines einzelnen neuen Objektes zu reorganisieren.

Die Suche erfolgt über den Vergleich des Anfrageobjekts mit den in den Knoten gespeicherten MBB. Sie startet am Wurzelknoten. Ist die Schnittmenge aus Anfrageobjekt und MBB nicht leer, dann muss in dem mit der MBB verknüpften Teilbaum weiter gesucht werden. Da Regionen nicht disjunkt sein müssen, können verschiedene Pfade im R-Baum auf das gleiche Objekt verweisen.

R$^+$-Baum Im Gegensatz zum R-Baum werden beim R$^+$-Baum [188] ausschließlich disjunkte Regionen in der Indexstruktur abgespeichert. Überlappt ein Objekt mit an-

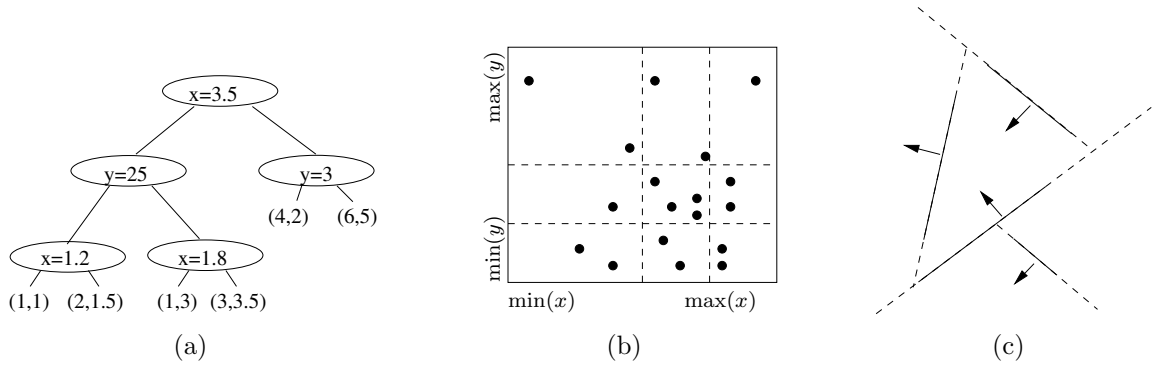


Abbildung 5.2: Index-Strukturen: (a) KD-Baum, (b) Grid-File-Regionen, (c) BSP-Baum-Regionen

deren Teilregionen, muss dieses durch geeignetes Zerlegen (*Clipping*) aufgeteilt werden. Die disjunkte Speicherung von Objekten erlaubt eine beschleunigte Punktanfrage und eine exakte Suche. Das Einfügen neuer Objekte und die Suche über regionenübergreifenden Objekten wiederum ist mit einem erheblichen Mehraufwand gegenüber dem R-Baum verbunden. Zudem wird durch die disjunkte Speicherung der Objekte zusätzlicher Speicherplatz belegt.

R*-Baum Einer der wesentlichen Schwachpunkte des R-Baums ist die Performance beim Einfügen neuer Objekte. In [27] beschreiben Beckmann et al. den R*-Baum als Erweiterung des R-Baumes. Dieser übernimmt die eigentliche Indexstruktur vollständig vom R-Baum. Die wesentlichen Änderungen beziehen sich auf das Einfügen neuer Objekte und die Aufspaltung von Teilregionen beim Überlauf. Das Löschen und die Suche nach Objekten werden nicht angepasst. Beim Überlauf eines Knotens im R*-Baum wird dieser nicht umgehend aufgespalten. Stattdessen wird zuerst in dem Knoten Platz geschaffen, indem ein Teil der mit diesem assoziierten Objekte gelöscht und an anderer Stelle im Baum wieder eingefügt wird. Dadurch muss der R*-Baum weniger häufiger aufgespalten werden. Zudem erhöht sich der Füllgrad, und die Tiefe des Baumes steigt weniger stark an, so dass Suchanfragen effizienter bearbeitet werden können. Für das Aufspalten wird die Einhaltung verschiedener Eigenschaften gefordert. Zum Beispiel gilt es, die Menge der Überlappungen zwischen Regionen zu minimieren, um bei späteren Suchanfragen weniger Teilbäume untersuchen zu müssen.

KD-Baum Bereits 1975 präsentierte Bentley den KD-Baum [29], einen multidimensionalen Binärbaum zur Speicherung von Datenpunkten. Die Blattknoten entsprechen dabei den k-dimensionalen Datenpunkten. Innere Knoten können als achsenparallele Splits der Hyperebene betrachtet werden. Im linken Teilbaum werden die Punkte gespeichert, die im linken Teil der geteilten Hyperebene liegen, und im rechten Teilbaum die von der rechten Hyperebene. Die wesentliche Herausforderung beim KD-Baum ist

die geeignete Wahl der Splitknoten. Eine ungünstige Aufteilung hat eine u.U. kostenintensive Suche im Fall von Anfragen zur Folge. In [151] wurde gezeigt, dass kompakte (dicke) Regionen zu besseren Ergebnissen führen als zum Beispiel längliche (dünne) Regionen. Zu den wesentlichen Vorteilen des KD-Baumes zählt der verhältnismäßig geringe Speicheraufwand. Da lediglich Punkte als Objekte zugelassen sind, werden diese nur einmalig im Baum gespeichert. Demgegenüber steht eine höhere Anfragekomplexität als bei anderen Indexstrukturen. Abbildung 5.2(a) zeigt ein Beispiel für einen KD-Baum mit sechs Datenpunkten.

Grid-File Beim Grid-File [168] wird der Raum durch achsenparallele Geraden in Zellen geteilt, welche ausschließlich Punkte aufnehmen können. Hierzu müssen die Grenzen für alle Dimensionen bekannt sein. Das Erzeugen der Zellen erfolgt sukzessive in Abhängigkeit ihres Füllgrades. Enthält eine Zelle zu viele Punkte, dann muss ein entsprechender Split durchgeführt werden. Unterschreitet der Füllgrad eine definierte Grenze, werden benachbarte Zellen zusammengefasst. Abbildung 5.2(b) zeigt die Regionenaufteilung im Fall des Grid-Files.

Weitere interessante Indexstrukturen sind zum Beispiel der Zell-Baum [95] und der BSP-Baum [75]. Der Zell-Baum ähnelt in der Struktur dem R-Baum, wobei anstelle der MBB konvexe disjunkte Polygone in den Knoten abgelegt werden. Damit ergeben sich ähnliche Probleme wie beim R-Baum. Um die Disjunktheit zu garantieren, müssen Polygone u.U. zerlegt werden, was wiederum in größeren Speicher- und Verwaltungskosten resultiert. Der BSP-Baum ist eine Erweiterung des KD-Baumes, wobei der Split der Hyperebene nicht parallel zu einer der Achsen erfolgen muss (Abbildung 5.2(c) zeigt die Regionenaufteilung beim BSP-Baum). Analog zum KD-Baum handelt es sich hierbei um eine Struktur, welche im Wesentlichen auf Punktobjekte beschränkt ist.

KD-Baum und Grid-File sind für die Integration in ein datenstromverarbeitendes System weniger geeignet, da lediglich Daten-Punkte verwaltet werden können. Zwar lassen sich Regionen auch durch entsprechende Transformationen [181] im KD-Baum bzw. im Grid-File ablegen, dies ist aber mit einem erhöhten Berechnungsaufwand, sowohl für das Einfügen als auch für die Suche verbunden. Beim KD-Baum kommt noch die kostenintensive Suche aufgrund der großen Baumtiefe hinzu (siehe Abbildung 5.2(a)).

Im Weiteren wird der R*-Baum nach [27] als Indexstruktur für Spatial-Join-basierte Operatoren eingesetzt. Aufgrund der hohen Performance im Fall von Anfragen und der Möglichkeit, auch komplexe geometrische Objekte ohne Transformation integrieren zu können, scheint dieser am besten geeignet für die Datenstromverarbeitung.

5.1.2 Raumbezogene Anfrageverarbeitung in *AnduIN*

Im folgenden Abschnitt wird die Integration raumbezogener Anfragen in *AnduIN* beschrieben [97]. Der Schwerpunkt ist dabei die Verarbeitung zweidimensionaler Geodaten (Latitude vs. Longitude), welche durch die Sensorknoten zur Verfügung gestellt werden.

Um das System möglichst einfach zu gestalten, liegt diesen ein geographisches Koordinatensystem zugrunde (d.h. Sensoren verfügen über keine Höhenangaben). Zusätzlich können auch zweidimensionale kartesische Koordinaten verarbeitet werden. Das kartesische Koordinatensystem findet hierbei im Wesentlichen bei der Verarbeitung nichtgeografischer Daten Verwendung, wie sie zum Beispiel bei der Verarbeitung von Sensordaten in Gebäuden auftreten. Aufgrund der notwendigen Genauigkeit und der hohen Dichte an Sensoren bietet sich bei diesem Szenario die Verwendung eines Bezugspunktes im oder nahe dem Gebäude an.

Im Folgenden soll auf die von *AnduIN* unterstützten Objekttypen, Formen und Operatoren mit räumlichem Bezug eingegangen werden. Die unterstützten Formen und Operatoren ermöglichen eine grundlegende Verarbeitung von Daten mit räumlichem Bezug.

5.1.2.1 Mobile vs. statische Objekte

Betrachtet man Sensoren als mögliche räumliche Objekte, so müssen prinzipiell die folgenden beiden Klassen unterschieden werden:

- *Statische Objekte:* Das Objekt (zum Beispiel ein Sensorknoten) ändert seine Position gegenüber dem Bezugssystem nicht.
- *Mobile Objekte:* Das Objekt bewegt sich im Raum und stellt selbstständig seine Positionsänderung (zum Beispiel via GPS) fest. Die veränderten Ortsangaben müssen dem System kontinuierlich mitgeteilt werden.

Die Verarbeitung von statischen und mobilen Objekten ist grundlegend verschieden. Im Fall der Verarbeitung statischer Objekte ist es ausreichend, dem System die Position des Objektes einmalig bekannt zu machen. Ist die Position aller Objekte, welche in einer Anfrage verarbeitet werden sollen, zum Initialisierungszeitpunkt der Anfrage bekannt und sind diese Objekte statisch, so können sämtliche für die räumliche Anfrageverarbeitung notwendigen Arbeitsschritte einmalig zum Ausführungszeitpunkt der Anfrage durchgeführt werden. Im Gegensatz dazu muss das System bei Anfragen über mobilen Objekten kontinuierlich mit den Positionsdaten *aller* mobilen Objekte versorgt werden. Zudem muss die Ergebnismenge kontinuierlich an die neuen Positionsangaben angepasst werden.

Statische Sensorknoten lassen sich in *AnduIN* im Metadaten-Katalog registrieren. Jeder Sensorknoten wird über einen eindeutigen Identifier registriert. Die entsprechenden CQL-Erweiterungen zum Verwalten von registrierten Sensoren finden sich im Anhang, Abschnitt C.5. Anfragen mit einem räumlichen Bezug können anschließend über den entsprechenden Identifier, welcher Teil des Datenstromes ist, auf die mit dem Knoten verbundenen Geoinformationen zurückgreifen.

Im Fall sich *bewegender Sensorknoten* ist eine Speicherung der Positionsangaben im Metadaten-Katalog nicht sinnvoll. Informationen über die aktuelle Position der Datenquelle sind Teil des Datenstromes. Dies heißt insbesondere, dass jeder Operator, welcher

diese Daten auswerten soll, „wissen“ muss, an welcher Position im Datenstrom sich die Geoinformationen befinden.

5.1.2.2 Geometrische Formen

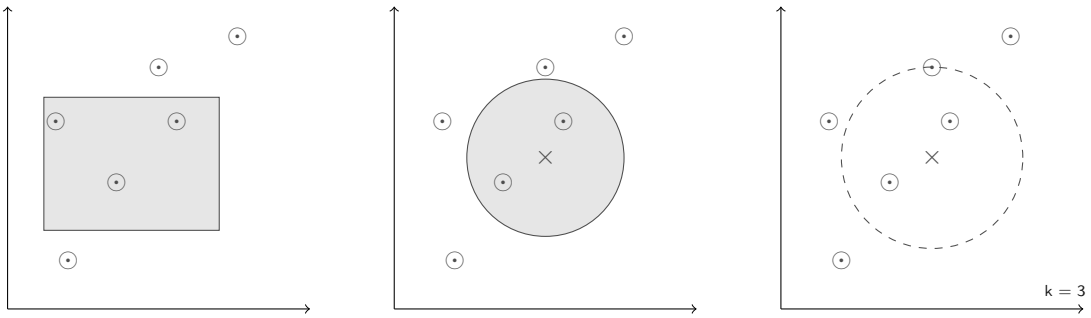
Im Allgemeinen sind bei der Analyse von Geoinformationen physische Objekte wie zum Beispiel Straßen, Flüsse, Wälder oder Landschaften von Interesse. Für die automatische Verarbeitung müssen diese Objekte als geometrische, vom System analysierbare abstrakte Formen abgebildet werden. In *AnduIN* stehen die folgenden abstrakten (zweidimensionalen) Formen zur Verfügung:

- *Punkte* erlauben die Beschreibung von Positionen im Raum. Sie besitzen dabei keine Ausdehnung. Die Positionen von Sensorknoten werden in *AnduIN* durch Punkte repräsentiert.
- *Linienzüge* sind Verbindungen von mindestens zwei Punkten. Die Verbindung zweier Punkte wird oftmals auch als Segment bezeichnet. Ein Linienzug ist eine Kette von sich an den Enden berührenden Segmenten. Mit Hilfe von Linienzügen können Objekte wie zum Beispiel Straßen oder Flüsse abgebildet werden. Sie verfügen selbst über keine räumliche Ausdehnung. Prinzipiell lässt sich dies aber durch Verwendung entsprechender Auflösungen simulieren.
- *Polygone* repräsentieren geometrische Objekte mit räumlicher Ausdehnung. Sie werden durch geschlossene Linienzüge abgebildet, d.h. der Startpunkt des Linienzuges entspricht zugleich dem Endpunkt. Mit Hilfe von Polygonen lassen sich Regionen wie zum Beispiel Wälder oder Seen definieren. In *AnduIN* beschränken wir uns auf einfache Polygone ohne Löcher.
- *Rechtecke* sind ein Spezialfall von Polygonen, welche gesondert betrachtet werden. Für die Definition von Rechtecken ist prinzipiell die Angabe zweier Eckpunkte ausreichend. Zudem existieren im Vergleich zu Polygonen effizientere Algorithmen für das Prüfen verschiedener Eigenschaften von Rechtecken (wie zum Beispiel das Prüfen auf Enthaltensein eines Punktes) als für Polygone.

Mit den beschriebenen geometrischen Objekten lassen sich die meisten realweltlichen räumlichen Objekte beschreiben. In traditionellen DBMS stehen zusätzlich oftmals noch Kreisbögen bzw. Kreise zur Verfügung. Kreisobjekte werden in *AnduIN* nicht explizit angeboten, aber über den *within distance*-Operator indirekt als Anfrageregion unterstützt.

5.1.2.3 Operatoren

Eine wichtige Klasse raumbezogener Anfragen sind Filter. Analog zum einfachen Filter für nichträumliche Daten werden Objekte, welche einer Filterbedingung nicht genügen, verworfen. Zu den räumlichen Filtern zählen die folgenden:



(a) Inside (b) Within Distance (c) k Nearest Neighbor

Abbildung 5.3: räumliche Filter-Operatoren in *AnduIN*

- Der *Inside*-Filter prüft, ob ein Objekt in einer definierten Region liegt (Abbildung 5.3(a)). Die Region kann dabei eine beliebige von *AnduIN* unterstützte geometrische Form sein.
- Beim *Within-Distance*-Filter wird getestet, ob sich ein Objekt in einem definierten Umkreis zu einem anderen Objekt oder einem beliebigen Punkt im Raum befindet (Abbildung 5.3(b)). Hierbei handelt es sich um einen Spezialfall des *Inside*-Filters, da als Region eine Kreisregion um einen zweiten Punkt angenommen wird.
- Der *k-Nearest-Neighbor*-Filter prüft, ob das geprüfte Objekt zu den k am nächsten liegenden Objekten bezüglich eines definierten Referenzpunktes gehört (Abbildung 5.3(c)).

Im Weiteren soll die Umsetzung der einzelnen Operatoren in *AnduIN* im Detail beschrieben werden. Im Fall sich nicht bewegender (statischer) Objekte kommt dabei für alle Operatoren ein *Whitelist*-Ansatz zum Tragen. Hierbei wird während der Initialisierungsphase der Anfrage eine Liste aufgebaut, welche die Identifikatoren aller Objekte enthält, die den entsprechenden Filter passieren. Die Laufzeitkomponente des Operators prüft anschließend lediglich, zu welchem Objekt ein Tupel gehört und ob dieses Objekt auf der Whitelist steht. Ist das der Fall, darf das Tupel den Operator passieren.

Inside-Operator Der Inside-Operator prüft, ob das Daten produzierende Objekt innerhalb einer definierten Anfrageregion liegt. Die Umsetzung des Inside-Operators erfolgt auf Grundlage eines Spatial Join mit dem Prädikat *contains*. Ein typisches Beispiel für die Anwendung des Inside-Operators wäre „Finde alle Sensoren, die sich im Wald befinden!“.

Während der Initialisierung der Anfrage werden alle Regionen, die eine Anfrage definieren in einen R*-Baum eingefügt. Im Fall statischer Objekte wird anschließend eine Whitelist erzeugt, indem für jeden einzelnen registrierten Sensor anhand des R*-Baumes überprüft wird, ob sich dieser innerhalb der in der Anfrage definierten Regionen befindet.

```
   Input : Anfrageregionen R = {r_1, r_2, ...}, registrierte Sensoren Nodes = {n_1, n_2, ...}
   Output : Whitelist W = {n_i, n_j, ...}
 1 RSB = createEmptyRStarTree();
 2 for r ∈ R do
 3     insertRegion(r, RSB);
 4 W = ∅;
 5 /* erzeuge Whitelist */
 6 for n ∈ Nodes do
 7     R_cand = isInRegion(n, RSB);
 8     isIn = TRUE;
 9     while isIn && R_cand ≠ ∅ do
10         r ∈ R_cand; R_cand = R_cand \ r;
11         if nodeIsInRegion(node, r) then
12             isIn = FALSE;
13     if isIn = TRUE then
14         W = W ∪ n;
15 return W;
```

Algorithmus 5.1 : Initialisierungsphase für statische Sensoren und mittels Schnitt definierte Anfrageregionen

Algorithmus 5.1 zeigt den Ablauf der Initialisierung für den statischen Fall. Der Inside-Operator unterstützt die Definition von Anfrageregionen mittels Schnitt und Vereinigung beliebiger Anfrageregionen. In Algorithmus 5.1 wurde sich der Einfachheit halber auf den Schnitt beschränkt.

Im Fall der Verarbeitung sich bewegender Objekte wird für *jedes* neu eintreffende Tupel auf Basis des R*-Baumes aufs Neue geprüft, ob es sich innerhalb der definierten Regionen befindet oder außerhalb. Zunächst werden hierzu alle MBB's bestimmt, in denen sich das Objekt (repräsentiert durch den Datensatz) möglicherweise befindet. Anschließend werden die entsprechenden Regionen im Detail extrahiert. In Abhängigkeit von der Form der Verknüpfung, dem Schnitt oder der Vereinigung der einzelnen Regionen wird entschieden, ob das Tupel den Operator passieren darf oder nicht. Dies entspricht in den wesentlichen Schritten dem Erzeugen der Whitelist aus Algorithmus 5.1.

Aufgrund der Verwendung des R*-Baumes ist es prinzipiell möglich, die Anfrageregionen zur Laufzeit anzupassen. Bei der statischen Variante muss hierzu die komplette Whitelist überarbeitet werden. Im Fall einer kontinuierlichen Prüfung von Tupeln ist das Hinzufügen von Anfrageregionen zum R*-Baum bzw. das Entfernen von Anfrageregionen aus dem R*-Baum jederzeit möglich. Des Weiteren sind bei der Auswertung, falls nötig, noch die Prädikate anzupassen.

k-Nearest-Neighbor-Operator Der k-Nearest-Neighbor-Operator erwartet neben der Definition der zu untersuchenden Objekte zusätzlich die Angabe eines Punktes im Raum bzw. eines Sensors, welcher als Pivot-Element dient. Im Fall der Verwendung eines Pivot-Sensors muss zusätzlich noch unterschieden werden, ob es sich beim verwendeten

Input : neuer Datensatz *tuple*, Koordinaten des Pivot-Elementes k_{pivot}, aktuelle (sortierte) Whitelist W
Output : $TRUE$ oder $FALSE$

```
1 k_n = extractGeoInf(tuple);
2 /*Koordinaten des maximal vom Pivot-Element entferntes Objektes*/;
3 k_max = maxDistance(W) ;
4 if distance(k_n, k_pivot) ≤ distance(k_max, k_pivot) then
5 |   W = (W \ k_max) ∪ k_n;
6 |   return TRUE;
7 return FALSE;
```

Algorithmus 5.2 : k-Nearest-Neighbor-Operator – Prüfen eines von einem sich bewegenden Sensor stammenden Tupels bei statischenm Pivot-Element

Sensor um ein statisches oder ein mobiles Objekt handelt.

Im Fall sich nicht bewegender Objekte und eines statischen Pivot-Punktes wird beim k-Nearest-Neighbor-Operator ebenfalls mit einer Whitelist gearbeitet. Diese umfasst die k am nächsten liegenden Elemente in sortierter Reihenfolge. Während der Initialisierungsphase werden die zum Pivot-Punkt k nächsten Elemente bestimmt und der Whitelist hinzugefügt. Bei der Behandlung sich bewegender Objekte muss während des Eintreffens neuer Datensätze für jeden Datensatz neu entschieden werden, ob das Daten produzierende Objekt zu den k-nächsten Nachbarn zählt. Zu diesem Zweck liegen die Elemente der Whitelist sortiert vor. Trifft ein neues Tupel ein, welches einem näheren Objekt als dem weitesten vom Pivot-Element entfernten Objekt der Whitelist zugeordnet werden kann, so werden die Positionsangaben aus diesem Datensatz an entsprechender Stelle in die Whitelist aufgenommen. Das bisher am weitesten vom Pivot-Element entfernte Element der Whitelist wird aus dieser verdrängt. Abbildung 5.2 zeigt den Algorithmus für das Prüfen eines von einem sich bewegenden Sensor stammenden Datensatzes auf Zugehörigkeit zur Whitelist bei statischem Pivot-Element.

Da im Fall sich bewegender Objekte zu Beginn nicht alle Objekte in der direkten Umgebung des Pivot-Elementes bekannt sind, kann es zu Beginn der Filterung zum Passieren von falsch-positiven Datensätzen kommen. D.h. Datensätze, welche prinzipiell nicht zu den k-nächsten Nachbarn zählen und somit den Operator nicht passieren dürften, passieren den Operator, solange keine Tupel von den tatsächlichen k-nächsten Nachbarn eingegangen sind.

Bei sich bewegendem Pivot-Sensor kann sich in Abhängigkeit von Bewegungsrichtung und -geschwindigkeit die Menge der nächsten Nachbarn im Verlauf der Analyse zwei aufeinander folgender Datensätze vollständig ändern. D.h. die „alte“ Liste der nächsten Nachbarn kann vollständig ungültig sein. Ein einfacher Lösungsansatz ist das Verwerfen aller bisherigen Nachbarn in der Liste bei einer Positionsänderung des Pivot-Elements und das anschließende Neuaufbauen einer aktuellen Liste (Algorithmus 5.3). Das größte Problem hierbei ist die erneute „Einschwing-Phase“ des Operators, welche in einer entsprechend schlechten Ergebnismenge resultiert. Im Fall einer hohen Update-Frequenz des Pivot-Elementes kann dieser Ansatz zu stark fehlerhaften Ergebnissen führen.

Input : neuer Datensatz $tuple$, Pivot-Sensor-ID id_p, aktuelle Koordinaten des Pivot-Elements k_{pivot}, aktuelle (sortierte) Whitelist W
Output : $\{k_{pivot}, TRUE \vee FALSE\}$

```
1 isIn = FALSE;
2 if tupleFromPivotObject(tuple, id_p) then
3     k'_pivot = extractGeoInf(tuple);
4     if k_pivot ≠ k'_pivot then
5         k_pivot = k'_pivot;
6         W = ∅;
7 else
8     isIn = staticNearestNeighborLookUp(tuple, p_koord, W) //analog Algorithmus 5.2;
9 return {k_pivot, isIn};
```

Algorithmus 5.3 : Nearest-Neighbor-Operator (bewegende Sensoren und Pivot-Element)

Within-Distance-Operator Der Within-Distance-Operator als Spezialfall des Inside-Operators arbeitet im Wesentlichen analog zu diesem. Aufgrund der Verwendung eines Pivot-Punktes und der Verwendung als Anfrageregion kann auf die Verwendung einer Indexstruktur verzichtet werden. Die Verarbeitung im Fall statischer Objekte erfolgt in ähnlicher Weise wie bei den anderen Operatoren. Zunächst werden alle Objekte im Umkreis des Pivot-Punktes bestimmt und in die Whitelist aufgenommen. Diese wird anschließend für die eigentliche Filterung verwendet.

Die Verarbeitung von Daten sich bewegender Sensoren gestaltet sich ähnlich einfach. Beim Eintreffen eines neuen Tupels wird berechnet, ob sich dieses innerhalb der vorgegebenen Distanz zum Pivot-Punkt befindet.

Die für eine deklarative Beschreibung räumlicher Anfragen notwendigen sprachlichen Erweiterungen von CQL finden sich in Anhang C.3.12.

5.1.3 Verarbeitung räumlicher Anfragen im WSN

Da es sich bei den implementierten räumlichen Operatoren im Wesentlichen um Filter handelt, sollten diese möglichst nah an den Datenquellen platziert werden, um so frühzeitig für eine Datenreduktion im Strom zu sorgen. In Analogie zu den für die einfachen Filter entworfenen Regeln, wurden entsprechende Regeln für räumliche Filter im logischen Optimierer umgesetzt.

Handelt es sich bei den in der Anfrage spezifizierten Datenquellen um Sampling-Operatoren im WSN, dann spricht generell nichts gegen eine Auslagerung der Filter-Operatoren in das WSN. Im Gegenteil, eine Filterung bereits auf den Sensorknoten kann zu einer drastischen Reduktion an zu sendenden Daten führen. Prinzipiell könnte sogar das Aufwachen und Messen von Werten bei Sensorknoten, welche nicht Teil der Anfragemenge sind, entfallen.

Die im Rahmen dieser Arbeit entwickelte In-Network-Lösung für räumliche Operatoren

beschränkt sich auf sich nicht bewegende Sensoren. Der implementierte Ansatz soll im Folgenden beschrieben werden. Auch wenn keine entsprechende Lösung für sich bewegende Operatoren vorliegt, so soll anschließend zumindest ein kurzer Überblick über entsprechende Umsetzungsmöglichkeiten gegeben werden, welche sich im Rahmen von *AnduIN* ergeben.

5.1.3.1 Statische Sensorknoten

Im Fall von sich nicht bewegenden Sensorknoten kann prinzipiell der Whitelist-Ansatz auch für die In-Network-Verarbeitung übernommen werden. Die Whitelist muss zu diesem Zweck einmalig auf der zentralen Instanz erzeugt und anschließend als Teil der Laufzeitumgebung auf die Sensorknoten verteilt werden. Hierbei stellt sich allerdings das Problem, dass die Whitelist-Erzeugung schon Teil der Operatorausführung ist. Gemäß der Anfrageverarbeitung in *AnduIN* (Kapitel 4) wird aber nach der physischen Optimierung der ausgewählte Plan direkt in die beiden Teilpläne zerlegt und im Anschluss daran zur Ausführung gebracht bzw. im WSN verteilt. Für die Verwendung einer Whitelist müssen daher an der initialen Anfrageverarbeitung entsprechende Anpassungen vorgenommen werden. Nachdem der optimale physische Plan für die Ausführung auf den einzelnen Komponenten zerlegt wurde, muss dieser auf vorhandene räumliche Operatoren hin untersucht werden. Werden diese auf nichtbewegliche Sensorknoten angewandt, so können anschließend die entsprechenden Whitelists erstellt werden. Die vollständige Filterliste kann dann als Teil der Laufzeitumgebung im WSN verteilt werden. In Abhängigkeit von ihrem eigenen Knotenidentifikator stellen die einzelnen Sensorknoten anschließend fest, ob die entsprechenden Tupel den betroffenen räumlichen Operator passieren dürfen oder nicht.

Die Integration der vollständigen Liste in die Laufzeitumgebung ist allerdings auch mit entsprechenden Problemen verbunden: (i) je nach Umfang der Whitelist geht Speicher in der Laufzeitumgebung verloren und (ii) das Verteilen eventueller (Anfrage-)Updates im WSN wird entsprechend teurer. Prinzipiell lassen sich die räumlichen Operatoren auch als *feedback basierte* Operatoren umsetzen (siehe Klassifikation Abschnitt 4.1), d.h. die Whitelist wird zur Laufzeit von der zentralen Komponente erstellt und anschließend im Netzwerk propagiert. Da aber auch in diesem Fall die vollständige Liste im WSN verteilt werden muss (jeder Knoten muss über eine Aktivierung bzw. Deaktivierung seinerseits informiert werden), ergibt sich gegenüber dem zu Beginn vorgestellten Ansatz kein wirklicher Kostenvorteil.

5.1.3.2 Mobile Sensorknoten

Das Problem sich bewegender Objekte (*Moving Objects*) ist ein hochaktuelles Forschungsthema [117, 183, 216]. Jenseits von drahtlosen Sensornetzwerken gibt es mittlerweile eine Vielzahl akkubetriebener Klein- und Kleinstgeräten (zum Beispiel Mobiltelefone, Smartphones, Notebooks etc.), deren aktuelle Position je nach Anwendung ausgewertet wird. Eine vollständige Betrachtung würde den Rahmen der Arbeit sprengen.

Daher beschränken sich die folgenden Ausführungen auf das Aufzeigen von Problemen, welche sich in diesem Zusammenhang ergeben.

Am einfachsten gestaltet sich die WSN-Integration des Within-Distance-Operators bei nichtbeweglichem Pivot-Element. Hierzu muss lediglich jeder Knoten des WSN „wissen“, wo sich das Pivot-Element befindet und in welchem Radius zu diesem eigene Tupel den Operator passieren dürfen. Die Auswertung kann somit auf jedem Operator unabhängig von anderen Sensorknoten stattfinden.

Bereits bei einem sich bewegenden Pivot-Element ist im Fall des Within-Distance-Operators eine Kommunikation zwischen den einzelnen Sensorknoten notwendig. Die Ortsveränderung des Pivot-Knotens muss den anderen Knoten im WSN bekannt gemacht werden. Noch komplexer gestaltet sich das Problem beim k-Nearest-Neighbor-Operator mit sich bewegendem Pivot-Element. Zusätzlich zum Aufenthaltsort des Pivot-Sensors müssen die Knoten sich noch Informationen über ihre eigene Nachbarschaftsbeziehung zu diesem austauschen. D.h. die notwendigen Informationen müssen in Teilen des WSN propagiert werden [114].

5.1.4 Verwandte Arbeiten

Zu den ersten Systemen, welche Daten mit räumlichem Bezug in großem Maßstab verwalteten, zählten die Geo-Informationssysteme. Mit der Einführung von GPS und anderen Lokalisierungssystemen war es prinzipiell jedem möglich, auf einfache Weise die Position von Objekten zu bestimmen und die Bewegung von Objekten zu verfolgen. Entsprechend sind in den letzten Jahren Erweiterungen zur Verarbeitung räumlicher Anfragen für die verschiedensten datenverarbeitenden Systeme entstanden.

Das von Oracle entwickelte 11g verwendet für die Darstellung von Geo-Objekten das objektrelationale Modell des Systems. Hierfür wurde ein entsprechender Geo-Datentyp eingeführt, welcher die Beschreibung von komplexen Polygonen, Kurven und Kreisen ermöglicht. Als Indexstrukturen setzt Oracle R-Bäume ein. Zu den wesentlichen von 11g unterstützten Operatoren zählen die in *AnduIN* integrierten Operatoren k-Nearest-Neighbor, Within-Distance sowie der in *AnduIN* nicht integrierte Relate-Operator (Relate untersucht die Beziehung von Objekten zueinander)[1]. In IBM's DB2 ist der *DB2 Spatial Extender* für die Verarbeitung von Objekten mit räumlichem Bezug verantwortlich. Geometrische Objekte werden in DB2 ebenfalls als Objekte beschrieben, was entsprechende Vererbungshierarchien ermöglicht. Als Indexstrukturen werden bei DB2 der *Grid-Index* und der *Geodätische Voronoi-Index* (als Teil des *Geodetic Extender*) eingesetzt. Beim Geodätischen Voronoi-Index wird die Erdoberfläche in (Voronoi-)Zellen zerlegt (ähnlich wie beim Grid-Index). Objekte werden dabei nicht durch Rechtecke, sondern durch ihren minimal umfassenden Kreis repräsentiert[2]. Neben diesen kommerziellen Systemen verfügen mittlerweile auch die bekanntesten Open-Source-DBMS wie

[1] Oracle Spatial – User's Guide and Reference (Stand 2010)
[2] IBM DB2 Spatial Extender and Geodetic Extender (Stand 2010)

zum Beispiel Mysql und PostreSQL über entsprechende Erweiterungen. Als Indexstrukturen verwenden beide Systeme R-Baum-Implementierungen.

Zum Zweck der Interoperabilität zwischen den Systemen wurde 2001 mit der ISO-Norm ISO/IEC 13249-3 SQL/MM eine entsprechende SQL-Erweiterung für den Umgang mit räumlichen Objekten in DBMS definiert [159, 196]. Die Spezifikation beschreibt unter anderem, wie räumliche Objekte als Werte darzustellen sind und welche Funktionen zur Verarbeitung bzw. zum Transformieren solcher Objekte vorhanden sein müssen.

Neben diesen Systemen zur Verarbeitung statischer Daten existieren mittlerweile auch erste Ansätze, Anfragen mit räumlichem Bezug in Datenstrom verarbeitende Systeme zu integrieren. Ein interessantes Projekt ist PLACE, welches das DSMS Nile um die Möglichkeit zur Formulierung raumbezogener Anfragen erweitert. Die Idee hinter PLACE [163, 164] ist, dass mobile Anwender Anfragen an die Serverkomponente stellen und diese Ergebnisse in Abhängigkeit ihres gegenwärtigen Aufenthaltsortes geliefert werden. Hierzu aktualisiert der Server kontinuierlich die Ergebnismenge für eine Nutzeranfrage im Fall von dessen Bewegung (zum Beispiel durch entsprechende Nachrichten über das Hinzufügen bzw. das Entfernen von Objekten in dessen Umgebung). PLACE* [221] ist eine verteilte Implementierung von PLACE.

Beim Projekt MobiEyes [82] übernimmt die Client-Anwendung einen großen Teil der Anfrageverarbeitung. Ziel der Client-Auslagerung ist dabei die Minimierung der Server-Last. Ein ähnliches Ziel verfolgt die Verwendung von bewegungsadaptiven Indexen. Dabei wird die voraussichtliche Bewegungsrichtung für jeden Client ermittelt und basierend auf dieser festgestellt, wann der Client voraussichtlich eine Zelle verlässt bzw. schneidet. Einen ähnlichen Ansatz zur Lastreduktion verfolgt der in [109] vorgestellte Ansatz. Hierbei wird für jedes sich bewegende Objekt eine „sichere Zone“ (rechteckige Umgebung um das Objekt) bestimmt, welche das Objekt in absehbarer Zeit nicht verlässt. So lange sich ein Objekt in dieser sicheren Zone befindet, müssen keine entsprechenden Änderungsnachrichten an den zentralen Server gesendet werden.

In [192] beschreiben die Autoren SPIX (*SPatial IndeX*), eine komplett im WSN verteilte Version eines R-Baumes. Jeder Sensorknoten verwaltet dabei einen MBB, der den Knoten selber und alle seine Kindknoten umfasst. Als Wurzel des R-Baumes dient die Basisstation. Der Zugang zum WSN ist wie bei *AnduIN* auf einen einzelnen Knoten beschränkt. Eine weitere Einschränkung ist die ausschließliche Integration von nicht beweglichen Sensorknoten. Der Aufbau des R-Baumes ist dabei in zwei Phasen unterteilt. Während der ersten, von der Basisstation via Broadcast-Nachrichten initialisierten Phase sammeln alle Knoten Informationen über ihre Nachbarn. Die zweite Phase dient der Bildung der R-Baum-Struktur, wobei der Aufbau beginnend bei den Blättern Richtung Wurzel erfolgt. Ein ähnlicher Ansatz wurde mit Peer-Tree in [63] präsentiert. Neben einer verteilten Version des R-Baumes präsentieren die Autoren zusätzlich auch einen k-Nearest-Neighbor-Ansatz, welcher auf dem verteilten Index basiert.

5.2 Adaptive Burst-Erkennung

Im folgenden Abschnitt soll ein Verfahren für die Erkennung von Bursts über nichtstationären Daten präsentiert werden. Im Gegensatz zum Problem der Ausreißererkennung, bei dem die Identifikation einzelner andersartiger Werte im Mittelpunkt steht, geht es bei der Burst-Erkennung um die Identifikation von abnormalem Verhalten über einem vorab nicht bekannten Zeitraum, welcher mehr als einen Einzelwert umfasst.

Die Identifikation von Bursts ist Teil einer Vielzahl von Steuer- und Analysesystemen. Im Bereich der Gebäudeautomation kann die Burst-Erkennung zum Beispiel zur Brand- bzw. Einbruchsdetektion eingesetzt werden. Beim Feststellen entsprechender Ereignisse werden anschließend automatisch Gegenmaßnahmen wie die Benachrichtigung von Feuerwehr oder Polizei eingeleitet. In beiden Fällen hat sowohl ein Nichterkennen des eingetretenen Ereignisses als auch ein falsches Auslösen negative Folgen. Ein weiteres Beispiel ist die Verkehrsüberwachung. Hierbei lassen sich mittels Burst-Erkennung automatisch Staus oder Unfälle erkennen. Eine Anwendung ohne Echtzeitanforderung ist die Auswertung astronomischer Beobachtungen [228]. Beispielsweise lässt sich die Burst-Erkennung zur Identifikation von Sternveränderungen einsetzen.

Zu den größten Herausforderungen im Zusammenhang mit der Identifikation von Bursts zählt deren möglichst frühzeitige und korrekte Erkennung. Werden Bursts zu spät oder gar nicht identifiziert, dann können eventuell notwendige Maßnahmen nicht mehr rechtzeitig eingeleitet werden. Kommt es hingegen zu einem häufigen Auslösen von *falsch-positiven* Anomalien, so können daraus eine hohen Systemlast und u.U. unnötig teure Folgemaßnahmen resultieren.

Die Erkennung von Bursts geschieht üblicherweise unter Verwendung von fensterbasierten Verfahren. Bei diesen wird geprüft, inwieweit die im aktuellen Fenster vorliegenden Daten von den Daten früherer Fenster abweichen. Für die Bestimmung der Abweichung wird sowohl auf statistische Verfahren als auch auf Verfahren aus dem Bereich der Signalanalyse oder der künstlichen Intelligenz zurückgegriffen. Die Detektion von Bursts über (nahezu) beliebigen Fenstergrößen wird als *elastic burst detection* bezeichnet. In [189, 232] präsentierten die Autoren hierfür einen Ansatz zur effizienten Detektion von Bursts mit variablen Zeitfenstern. Mit der als *shiftet aggregation tree* bezeichneten Datenstruktur wurde in [229] eine effiziente und einfach zu implementierende Lösung zur Burst-Detektion über Datenströmen vorgestellt. Aufgrund der verwendeten Schwellwertstrategie ist der vorgestellte Ansatz jedoch nur für stationäre Daten geeignet, deren zeitlicher Verlauf keine Trends oder Perioden aufweist. Datenströme (insbesondere bei längerer zeitlicher Überwachung) können allerdings sowohl saisonale Schwankungen als auch Trends enthalten. Betrachtet man zum Beispiel Außensensoren für Temperatur oder Luftfeuchtigkeit, so unterliegen diese systematischen Schwankungen in Abhängigkeit von der Tages- oder Jahreszeit. Diese können zu entsprechenden falsch-positiven Alarmen führen.

Im Folgenden soll ein Ansatz zur Burst-Detektion in Datenströmen präsentiert werden, welcher sowohl für stationäre als auch für nichtstationäre (trend- bzw. saisonbehaftete) Daten geeignet ist. Zunächst werden hierzu in Abschnitt 5.2.1 notwendige Grundlagen

(Datenstrukturen und Algorithmen) vermittelt. Anschließend wird in Abschnitt 5.2.2 eine auf der Zeitreihenvorhersage basierende Anpassung am vorgestellten Basisalgorithmus vorgenommen. In Abschnitt 5.2.3 werden die Umsetzung der vorgeschlagenen Lösung als In-Network-Operator und dessen Integration in *AnduIN* beschrieben. Abgeschlossen werden die Betrachtungen zum Problem der Burst-Erkennung mit einem kurzen Überblick über Vorarbeiten auf diesem Themengebiet.

5.2.1 Vorbetrachtungen

Ein einfacher Ansatz zur Identifikation von Bursts beliebiger Länge ist die Prüfung aller möglichen gleitenden Teilfenster beginnend ab dem Zeitpunkt t. Bei einem Datenstrom mit w Elementen und k verschiedenen Teilfenstergrößen wären somit $O(kw)$ Operationen notwendig, was im Fall von kontinuierlichen Datenströmen nicht praktikabel ist.

Eine effizientere Verarbeitung erlaubt die in [229] vorgestellte Aggregationspyramide (*aggregation pyramid*) bzw. der daraus abgeleitete Aggregationsbaum (*shifted aggregation tree*). Im Folgenden sollen diese Datenstrukturen genauer beschrieben werden, da sie die Basis der hier vorgestellten Lösung bilden.

5.2.1.1 Aggregationspyramide

Eine Aggregationspyramide ist eine Datenstruktur über w Stromelementen mit w Ebenen. w bezeichnet die Anzahl der Datenelemente im betrachteten Fenster. Ebene 0 der Aggregationspyramide enthält die w Elemente des Datenstromes und entspricht einem gleitenden Fenster der Länge w. Ebene 1 setzt sich aus $w-1$ Elementen zusammen, welche Aggregate aus jeweils zwei benachbarten Elementen der Original-Daten (Ebene 0) enthält. Der weitere Aufbau der Aggregationspyramide ist rekursiv definiert. Die $w-i$ Elemente der Ebene i lassen sich wie folgt aus den Elementen der darunter liegenden Ebene ableiten:

$$c(i,t) = Agg(c(i-1,t), c(0,t+i))$$

wobei $c(i,t)$ den aggregierten Wert der Zelle auf Ebene i zum Zeitpunkt t beschreibt. Agg bezeichnet die verwendete Aggregationsfunktion. Das Element auf Ebene $w-1$ enthält somit ein Aggregat über alle Elemente der Zeitreihe im Basisfenster. Abbildung 5.4 zeigt eine Aggregationspyramide, deren Fenster 8 Elemente des Datenstromes überdeckt.

Eine so aufgebaute Pyramide weist mehrere interessante Eigenschaften auf. So bildet zum Beispiel jede Zelle $c(i,t)$ die Spitze einer Teilpyramide, deren Fenster zum Zeitpunkt t startet und $i+1$ Elemente im Basisfenster überdeckt. Diese Eigenschaft lässt sich für die Detektion von Bursts ausnutzen. Überschreitet eine Zelle der Aggregationspyramide einen definierten Schwellwert, so bedeutet dies, dass in der von dieser Zelle überdeckten Teilpyramide ein Burst aufgetreten ist. Für eine genauere Beschreibung des Bursts muss anschließend die überdeckte Teilpyramide überprüft werden.

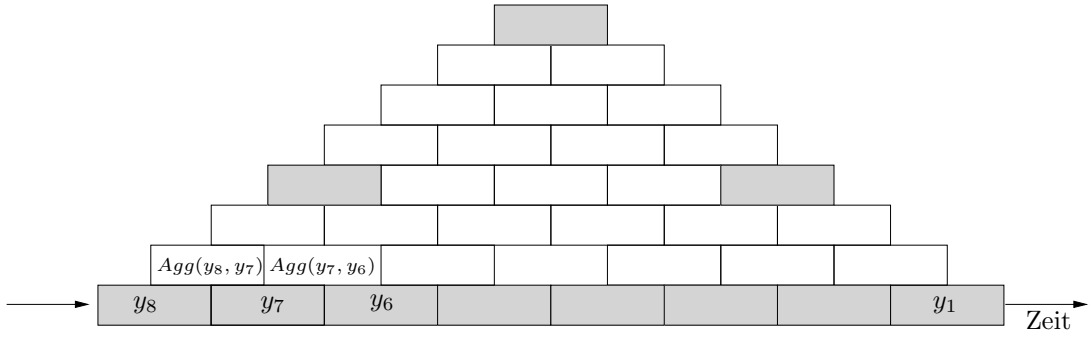


Abbildung 5.4: Aggregationspyramide mit Fenstergröße 8 und einem 3-Ebenen-Aggregationsbaum (hellgrau)

5.2.1.2 Aggregationsbaum

Das Einfügen neuer Werte in die Aggregationspyramide und die anschließende Suche ist mit erheblichen Kosten verbunden. Aus diesem Grund präsentierten die Autoren in [229] mit dem Aggregationsbaum eine effiziente Datenstruktur, dessen Knoten einer Teilmenge der Zellen der Aggregationspyramide entsprechen. Die Blattknoten des Aggregationsbaumes repräsentieren die Ebene 0 der Aggregationspyramide, d.h. die Werte in den Blättern entsprechen den Datenelementen des Stromes. Ein Knoten auf Ebene i des Aggregationsbaumes repräsentiert das Aggregat seiner Kindknoten auf Ebene $i-1$. Die in Abbildung 5.4 hellgrau eingezeichneten Elemente entsprechen den Knoten eines möglichen Aggregationsbaumes der Höhe 3, welcher sich aus der dargestellten Aggregationspyramide ableiten lässt.

Jede Datenstrom-Teilsequenz der Länge l mit $l \leq \omega_i - b_i + 1$ wird im Aggregationsbaum durch einen Knoten auf Level i überdeckt. ω_i bezeichnet hierbei die Größe des Fensters auf Ebene i und b_i ist der Versatz auf Ebene i. Jedes Teilfenster im Aggregationsbaum $\leq \omega_i - b_i$ wird somit durch mindestens einen Knoten auf Ebene i vollständig überdeckt. Folglich kann jeder Burst der Länge $\leq \omega_i - b_i$ aufgefunden werden.

Für das Finden eines Bursts innerhalb des Aggregationsbaumes ist es ausreichend zu prüfen, ob im Zeitraum $[t-b_i+1, t]$ ein Teilfenster der Größe k mit $\omega_{i-1} - b_{i-1} + 2 \leq k \leq \omega_i - b_i + 1$ den Schwellwert überschreitet. Im Fall einer Schwellwertüberschreitung muss eine detaillierte Suche im Teilbaum unterhalb des betroffenen Knotens vorgenommen werden.

Eines der wesentlichen Probleme der Datenstruktur ist die Wahl der Zellen, welche im Baum abgebildet werden. Je dünner der Aggregationsbaum besetzt ist, desto weniger Aktualisierungen müssen beim Einfügen eines neuen Datensatzes durchgeführt werden. Umso aufwendiger ist aber die Detektion von Bursts, da entsprechend viele Teilbäume unterhalb eines Knotens untersucht werden müssen. Bei einem dicht besetzten Baum hingegen fallen die Kosten für die Detektion eines Bursts geringer aus. Dafür ist aber der Aufwand für die Aktualisierung der Zellen beim Einfügen eines neuen Datensatzes entsprechend hoch. In [228] beschreibt der Autor mit dem State-Space-Algorithmus einen Ansatz, welcher je nach Anwendungsfall die beste Datenstruktur ermittelt. Der

Algorithmus versucht, die verschiedenen möglichen Ausprägungen unter Anwendung von Regeln in einen nahezu optimalen Aggregationsbaum zu überführen.

5.2.2 Nichtstationäre Burst-Erkennung

Bereits in [228] argumentiert der Autor, dass die Qualität der Burst-Detektion von der Wahl des Schwellwertes abhängt. Dies ist eine der wesentlichen Schwachstellen des präsentierten Verfahrens. Wird der Schwellwert zu groß gewählt, so werden unter Umständen kleinere Bursts nicht gefunden. Ist der Schwellwert hingegen zu klein gewählt, so kann dies zur Folge haben, dass auch kleinste Schwankungen fälschlicherweise als Bursts identifiziert werden.

Neben der Möglichkeit der manuellen Definition eines geeigneten Schwellwertes findet häufig die Standardabweichung der im Fenster enthaltenen Werte als Schwellwert Verwendung. Diese Herangehensweise weist allerdings zwei grundlegende Probleme auf: (i) Die Standardabweichung ist nicht stabil bezüglich einzelner Ausreißer (aufgrund der Abhängigkeit vom Mittelwert), d.h. ein einzelner Wert kann zu einem unverhältnismäßig großen Anstieg der Standardabweichung führen und (ii) passt sie sich verhältnismäßig langsam an nichtstationäre Daten an, die beispielsweise trendbehaftet sind oder saisonalen Schwankungen unterliegen. Während dem Problem der Stabilität durch entsprechende Transformationen entgegengewirkt werden kann [102], bleibt das zweite Problem bisher unberücksichtigt.

Das im Folgenden beschriebene Verfahren der adaptiven Burst-Erkennung [123,127,174] versucht daher, Trends und Schwankungen als Teil der Datenreihe zu erkennen und passt die Schwellwerte, welche der Burst-Erkennung dienen, entsprechend an. Für die Erkennung wird das Verfahren der exponentiellen Glättung eingesetzt. Bei der exponentiellen Glättung (*exponential smoothing*) handelt es sich um ein relativ einfaches, aber robustes Verfahren zur Vorhersage von Zeitreihen. Es wird in einer Vielzahl von Bereichen eingesetzt wie zum Beispiel bei der Bedarfsvorhersage in der Lagerhaltung [80]. Bei der exponentiellen Glättung werden prinzipiell drei Arten unterschieden: (i) *single exponential smoothing*, (ii) *second exponential smoothing* (Holt-Verfahren) und (iii) *third exponential smoothing* (Holt-Winters-Verfahren) [34]. Die einzelnen Verfahren unterscheiden sich dabei im Wesentlichen bezüglich ihrer Komplexität und der Qualität der Vorhersagen. Eine ausführliche Darstellung der Verfahren ist in Anhang A zu finden.

Prinzipiell können für die Datenvorhersage und damit die Erkennung von Trends beliebige Vorhersageverfahren eingesetzt werden (zum Beispiel andere ARMA-Modelle). Die exponentielle Glättung zeichnet sich unter diesen durch ihren relativ geringen Berechnungsaufwand aus. Diese Eigenschaft prädestiniert sie insbesondere für die Analyse über Datenströmen und erlaubt eine Auslagerung des vorgestellten Verfahrens auf die Sensorknoten.

Die verschiedenen Vorhersageverfahren der exponentiellen Glättung hängen von unterschiedlichen Gewichten ab, welche die Trends bzw. Perioden in dem analysierten Datenstrom widerspiegeln. Da zu Beginn der Burst-Erkennung keinerlei Wissen über

```
   Input : Datenwert y, aktueller Aggregationsbaum aggTree
   Output : Burst (Start- und Endzeitpunkt)
1  insertIntoAggrTree(aggTree, y);
2  if Training then
3      param = getForecastParam(aggTree) // ermittle Parameter für die Vorhersage;
4  else
5      F_{t+1} = computeForecast(aggTree, param) // berechne Vorhersage;
       thresh = computeThreshold(F_{t+1}) //berechne neuen Schwellwert ;
6      burst = burstDetection(aggTree, thresh) // nach [228];
7      return burst;
```

Algorithmus 5.4 : Algorithmus zur Burst-Erkennung

den Datenstrom vorhanden ist, müssen die Gewichte zunächst während einer initialen Trainigsphase bestimmt werden. Während dieser Phase ist eine Burst-Erkennung unter Verwendung von vorhergesagten Werten nicht möglich. An dieser Stelle muss entweder auf eine Erkennung von Bursts verzichtet werden oder es muss auf eine Überwachung mittels des Verfahrens für stationäre Daten nach [228] zurückgegriffen werden.

Die während der initalen Trainingsphase erlernten Werte dienen für alle zukünftigen Vorhersagen als Parameter. Prinzipiell sollte im weiteren Verlauf eine Adaption der Parameter durchgeführt werden, da sich die Verteilung der Daten ändern kann. Diese ist gegenwärtig aber nicht vorgesehen. Es sollte aber auf einfache Weise möglich sein, den Datenstrom parallel zur laufenden Auswertung auf die Gültigkeit der aktuellen Parameterwerte hin zu prüfen und nötigenfalls diese zu aktualisieren. Hierbei muss allerdings auf eventuell laufende Analysen geachtet werden, da eine Apassung währenddessen nicht ohne weiteres möglich ist.

Nach der initialen Phase werden die Schwellwerte für die Burst-Erkennung auf Basis der durch die Vorhersage ermittelten Werte bestimmt. D.h. anstatt die Standardabweichung über den Original-Datenstromelementen zu ermitteln, wird der Schwellwert auf Basis der vorhergesagten Werte bestimmt. Die Vorhersage von Werten erfolgt basierend auf den Datenelementen des Datenstromes, also auf Blattebene des Aggregationsbaumes. Für die einfache Glättung und die doppelte Glättung wurden die beiden nachstehenden Ansätze gewählt:

- *Einfache Glättung*: Für jeden neu eintreffenden Datenpunkt wird eine 1-Schritt-Vorhersage durchgeführt. Der so vorhergesagte Wert wird für die Schwellwertberechnung herangezogen.
- *Doppelte Glättung*: Auf Basis der letzten k-Werte werden der Mittelwert und der Anstieg der Zeitreihe ermittelt. Mit Hilfe dieser Werte kann eine entsprechende Vorhersage für die nächsten i-Werte getroffen werden. Die vorhergesagten Werte dienen im Anschluss der Schwellwertermittlung.

Die Vorhersagequalität sinkt umso mehr, je weiter der betrachtete Zeitpunkt in der Zukunft liegt. Aus diesem Grund werden lediglich kurze Vorhersageintervalle verwendet. Wurden alle vorhergesagten Werte zur Schwellwertberechnung herangezogen, dann

wird eine neue Approximation unter Verwendung der aktuellen Daten durchgeführt. Da eine Vorhersage von Werten basierend auf Daten, die während eines Bursts gewonnen werden, zu fehlerhaften Vorhersagen führen würde, wird die Vorhersage bis zum Ende eines erkannten Bursts ausgesetzt. Nachdem das Ende des Bursts festgestellt wurde, müssen bis zu k neue Werte für die nächste Vorhersage gesammelt werden. Erst dann ist ein erneutes Vorhersagen sinnvoll. Algorithmus 5.4 zeigt noch einmal die wesentlichen Schritte zur adaptiven Burst-Erkennung. In Abschnitt 6.2.2.2 wird das hier vorgestellte Verfahren evaluiert.

5.2.3 Burst-Erkennung im WSN

Die Wahl eines geeigneten Aggregationsbaumes ist ausschlaggebend für die Performance des Verfahrens. Ist der verwendete Baumtyp ungeeignet, so werden Einfüge- bzw. Suchaufgaben entsprechend teuer. Diese Tatsache sollte insbesondere im Fall der In-Network-Verarbeitung berücksichtigt werden, da die Ressourcen zur Berechnung stark beschränkt sind. Prinzipiell sollte der in [228] vorgestellte State-Space-Algorithmus die beste Baumstruktur finden. Allerdings handelt es sich hierbei um ein kostenintensives Optimierungsverfahren, dessen Auslagerung auf die Sensorknoten nicht sinnvoll ist. Aus diesem Grund wird der Burst-Operator (ähnlich wie die räumlichen In-Network-Operatoren) in zwei Teile aufgeteilt: (i) eine initiale Komponente, welche auf der zentralen Instanz die beste Baumstruktur ermittelt und (ii) eine ausführende Komponente, welche als Teil der Laufzeitumgebung im WSN propagiert wird. Der WSN-Teil kann als einfacher In-Network-Operator umgesetzt werden, welcher vollständig auf den Knoten verarbeitet wird. Eine spätere Kommunikation mit anderen Systemkomponenten ist nicht nötig.

Zum Zweck der Anfrageoptimierung müssen die Kosten für die Ausführung des Burst-Operators bestimmt werden. Darüber hinaus muss der Speicherbedarf des Operators approximiert werden. Prinzipiell lassen sich die für das Einfügen eines Datensatzes notwendigen Operationen und die Anzahl der im Mittel notwendigen Operationen für die Burst-Detektion aus der Baumstruktur ableiten. Zum Zeitpunkt der Anfrageoptimierung ist diese Struktur jedoch noch nicht im Detail bekannt (lediglich die Anzahl der Blätter, welche der Fensterbreite entspricht). Für die Lösung dieses Problems sind drei Ansätze möglich:

- Die Ermittlung des für die Ausführung verwendeten Aggregationsbaumtyps wird während der Planenumerationsphase durchgeführt, so dass anschließend die exakte Struktur während der Bewertung zur Verfügung steht.
- Die Bewertung des Aggregationsbaumes erfolgt immer unter der Annahme des am dichtesten besetzten Baumes für das definierte Problem. Hierbei handelt es sich immer um einen Binärbaum mit w-Blättern.
- Es wird ein Aggregationsbaum „mittlerer“ Dichte angenommen. Hierfür müssen sowohl die Baumhöhe als auch die Anzahl der Kinder pro Knoten vorab definiert

werden (zum Beispiel auf Basis von Erfahrungswerten).

Der Nachteil des ersten Ansatzes ist die Vermischung von Kostenbewertung und Anfrageausführung, was im Fall einer hohen Anzahl von möglichen Ausführungsplänen zu einer entsprechenden Verteuerung dieser Phase führen kann. Der zweite Ansatz, welcher die teuerste Baumstufe annimmt, kann u.U. zu einer Verschiebung der Kosten zugunsten einer zentralen Ausführung führen. Je größer die Anzahl der Knoten ist, desto höher sind die Einfügekosten. Der vollständig besetzte Aggregationsbaum verursacht somit die höchst möglichen Kosten für den Operator. Entsprechend hoch ist auch der Speicherbedarf, denn neben den Originalwerten in den Blättern sind zusätzlich $w - 1$ innere Knoten im Baum notwendig. Beim letzten Ansatz stellt die Spezifikation einer günstigen Baumstruktur die größte Herausforderung dar. Hier kann allerdings auf entsprechende Erfahrungswerte zurückgegriffen werden, so dass dieser Ansatz die besten Ergebnisse im Bezug auf Genauigkeit und Performance verspricht.

5.2.4 Verwandte Arbeiten

Das Problem der Ausreißer- bzw. Burst-Erkennung findet sich in einer Vielzahl von Anwendungen wieder. Neben der Bereinigung von Sensordaten zählen hierzu auch die Überwachung der TCP-, Web-, Datei-System- oder Festplatten-Last [210,213]. Oftmals steht dabei das Erkennen von Engpässen oder Angriffen im Mittelpunkt. Interessant ist auch das von Kleinberg in [129] betrachtete Problem der Erkennung von Bursts in Textdatenströmen (zum Beispiel E-Mails oder News). Kleinberg widmet sich dabei der Erkennung von Worthäufigkeiten in Textdatenströmen und deren zeitlicher Veränderung. So lassen sich zum Beispiel plötzlich auftauchende Themenänderungen erkennen.

Das Problem der Burst-Erkennung ist eng verwandt mit dem Problem der Ausreißerdetektion (*outlier detection*) [124]. Prinzipiell lassen sich Verfahren zur Erkennung von Bursts bzw. Ausreißern in zwei Klassen einteilen: überwachte und nichtüberwachte Verfahren. Erstere versuchen anhand von kategorisierten Trainingsdaten Charakteristika zu erlernen (*supervised learning*). Das erlernte Wissen wird anschließend für die Erkennung verwendet. Entsprechende Verfahren basieren zum Beispiel auf künstlichen neuronalen Netzen [83] oder verwenden Techniken des maschinellen Lernens [139,187]. Zweitere kommen ohne das Zuführen von externem Wissen aus (*unsupervised learning*). Bei diesen Verfahren werden oftmals statistische Eigenschaften wie zum Beispiel die Distanz oder die Dichte von Daten bzw. Datenmengen in Relation zueinander gesetzt. Bei den distanzbasierten Verfahren [26,130,131] wird geprüft, ob der Abstand eines Wertes zu seinen Nachbarwerten im Datenstrom einen definierten Schwellwert überschreitet. Ist dies der Fall, so handelt es sich um einen Ausreißer bzw. Burst. Im Allgemeinen finden distanzbasierte Verfahren lediglich globale Ausreißer. Im Gegensatz dazu betrachten dichtebasierte Verfahren, wie zum Beispiel das in [40] beschriebene, Punkte genau dann als Ausreißer, wenn sich deren lokale Dichte signifikant von der ihrer Nachbarn unterscheidet. Einen umfangreichen Überblick über verfügbare Verfahren aus den Bereichen Burst- bzw. Ausreißererkennung liefert [108].

Bei dem von Zhu und Shang in [232] präsentierten Verfahren zur Detektion von Bursts handelt es sich ebenfalls um einen distanzbasierten Ansatz. Es war dabei die erste Lösung, welche für die Detektion von Bursts auf Wavelets setzte. Die oben verwendete Arbeit [229] erweiterte diese Ansatz um die vorgestellten Datenstrukturen.

5.3 Quantitatives Frequent Pattern Mining

Als letzter Operator soll in diesem Kapitel stellvertretend ein Verfahren zum Finden häufiger Muster in Datenströmen beschrieben werden. Das Finden häufiger Muster (*frequent pattern mining*) ist Grundlage für eine Vielzahl von Data-Mining-Problemen (zum Beispiel für die Korrelationsanalyse oder das Finden von Sequenzen oder Perioden innerhalb eines Datenstromes). Die populärste Anwendung für das Frequent Pattern Mining ist das Assoziation Rule Mining, bei welchem aus Transaktionen Werte extrahiert und in Relation zueinander gesetzt werden. Eine typische Anwendung ist die Analyse von Supermarktransaktionen, bei der das Kaufverhalten von Kunden untersucht wird. Ziel ist dabei die Detektion von Regeln, die das Kaufverhalten einer repräsentativen Menge von Kunden widerspiegeln. Anhand der Regeln können anschließend die Verkaufsprozesse optimiert werden. Eine mögliche Regel wäre beispielsweise „Bier → Chips (10%)“, welche besagt, dass 10 Prozent aller Kunden, die Bier gekauft haben, auch Chips kauften.

Wie die meisten Data-Mining-Verfahren, so wurde auch das Frequent Pattern Mining ursprünglich für statische Datenbestände entwickelt. In den letzten Jahren wurden diese Verfahren auf das Datenstrom-Paradigma übertragen [101, 152]. Ein interessantes Beispiel für die Anwendung des Assoziation Rule Mining über Datenströmen ist in [98] beschrieben. Die Autoren versuchen dabei, fehlende Messwerte in einem Sensordatenstrom anhand zuvor abgeleiteter Assoziationsregeln zu ergänzen.

In vielen Anwendungen liegen die Daten zur Analyse nicht in Form kategorischer Attribute vor, sondern als quantitative Attribute. So messen Sensoren im Allgemeinen Werte aus einem stetigen Wertebereich. Die Übertragung existierender Frequent-Pattern-Mining-Verfahren auf solche Daten führt hierbei oftmals nicht zu den gewünschten Ergebnissen, und der resultierende Berechnungsaufwand ist nicht vertretbar. Basierend auf dieser Feststellung entwickelten die Autoren in [193] erstmals ein Verfahren für das Finden von Regeln, welches sowohl mit kategorischen Attributen als auch mit quantitativen (mengenwertigen) Attributen umgehen kann. Das vorgestellte Verfahren bildet dabei quantitative auf kategorische Attribute ab. Die quantitativen Attribute werden im Wesentlichen auf Intervalle abgebildet, welche eine Menge von numerischen Werten aufnehmen können. Auf Basis dieser Definition präsentierten die Autoren einen angepassten Apriori-Algorithmus, welcher den Suchraum zerlegt (diskretisiert) und anschließend durch Kombination der Teilintervalle Regeln findet. Der Apriori-Ansatz weist allerdings zwei grundlegende Probleme auf, welche ihn für die Anwendung über Datenströmen disqualifizieren: (i) Für das Finden der Items werden alle möglichen Kombinationen der Teilintervalle geprüft und (ii) ist das Apriori-Verfahren selbst aufgrund des

wiederholten Kombinierens von (k-1)-Itemsets zu k-Itemsets nicht für die Anwendung auf Datenströmen geeignet.

In den letzten Jahren sind einige Verfahren zum Finden häufiger quantitativer Itemsets entwickelt worden. Bei den vorgestellten Lösungen handelt es sich aber zumeist um Abwandlungen des Apriori-Ansatzes [193], so dass eine Analyse über Datenströmen nicht effizient möglich ist. Im weiteren Verlauf dieses Abschnitts soll daher ein geeignetes Verfahren für das quantitative Frequent Pattern Mining über Datenströmen präsentiert werden. Zunächst werden hierzu notwendige Definitionen vorgenommen. Anschließend soll mit dem *Frequent Pattern Partitioning Stream* (FP^2-Stream) [128] ein auf dem FP-Tree von Han et al. [101] basierender Lösungsansatz präsentiert werden. Danach wird die Integration des entwickelten Verfahrens in *AnduIN* beschrieben. Abgeschlossen wird dieser Abschnitt mit einem Überblick über Vorarbeiten zum Thema Frequent Pattern Mining im Allgemeinen sowie zum quantitativen Frequent Pattern Mining im Speziellen.

5.3.1 Vorbetrachtungen

Zunächst müssen verschiedene Begrifflichkeiten eingeführt werden, welche für das Verständnis des im Weiteren präsentierten Verfahrens notwendig sind.

Im Folgenden bezeichne A_i ein Attribut im Datenstrom und y einen einzelnen Attributwert. $A_i\langle y\rangle$ entspreche dem zum Attribut A_i gehörenden Wert y. Der Ausdruck $A_i(l,r)$ bezeichne ein Item über dem Attribut A_i. Das Item ist entweder ein *quantitatives Attribut* mit einem Wert im Intervall $[l,r]$ oder ein *kategorisches Attribut* mit dem Wert $l=r$.

$\mathcal{I}$ bezeichne die Menge aller Items. Die Menge $X=\{A_1(l,r),...,A_k(l',r')\}\subseteq\mathcal{I}$, repräsentiert ein Itemset (oder auch k-Itemset), wobei die A_i mit $1\leq i\leq k$ paarweise disjunkt sind.

D bezeichne eine Menge von Transaktionen. Eine Transaktion $d_{\Delta t}\in D$ ist eine Menge von Attributwerten $A_i\langle y\rangle$ im Zeitintervall Δt. Jede Transaktion $d_{\Delta t}=\{A_i\langle y\rangle,...,A_j\langle y'\rangle\}$ kann auf ein Itemset $\mathcal{I}$ abgebildet werden, d.h. für jeden Wert $A_i\langle y\rangle$ existiert ein Item $A_i(l,r)$ in $\mathcal{I}$ mit $y\in[l,r]$.

Die Häufigkeit $freq(X,D)$ eines Itemsets X entspricht der Anzahl der Transaktionen $d_{\Delta t}\in D$ im Zeitintervall Δt, die auf das Itemset abgebildet werden können. Der Support $supp(X,D)$ eines Itemsets ist definiert als der prozentuale Anteil an Transaktionen $d_{\Delta t}\in D$, welche auf das Itemset abgebildet werden können. Üblicherweise sind nur häufige Itemsets von Interesse. Ein Itemset wird als häufig bezeichnet, wenn dessen Häufigkeit einen vordefinierten Schwellwert *minsup* überschreitet.

Signifikanz und Informationsdichte Ziel des quantitativen Frequent Itemset Mining ist das Finden von zusammen auftretenden Items, deren Informationsgehalt sich *signifikant* von dem aller anderen Items unterscheidet. Ein quantitatives Item wird genau dann als signifikant bezeichnet, wenn dessen Informationsdichte größer ist als die

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
A_1	22	22.5	-	22.5	21.5	21.5	21	-	22	22.5
A_2	100	-	100	-	0	0	-	100	-	100

Tabelle 5.2: Beispiel Transaktionen

itemset	$freq$	$supp$
$\{A_1(20, 21.5]\}$	3	0.3
$\{A_1(20, 23]\}$	8	0.8
$\{A_2(0, 50]\}$	4	0.4
$\{A_2(50, 100]\}$	4	0.4
$\{A_1(20, 21.5], A_2(0, 50]\}$	2	0.2
$\{A_1(20, 23], A_2(0, 50]\}$	2	0.2
$\{A_1(20, 23], A_2(50, 100]\}$	2	0.2

Tabelle 5.3: Beispiel Itemsets

Informationsdichte der alternativen Items. Die Informationsdichte eines Items $A_i(l, r)$ ist dabei wie folgt definiert:

$$density(A_i(l, r)) = \frac{freq(A_i(l, r))}{|l - r|} \tag{5.1}$$

Wobei $|l - r|$ der Distanz zwischen den beiden Intervallgrenzen des Items entspricht. Entsprechend dieser Definition wird ein Itemset $\mathcal{I}$ genau dann als signifikant bezeichnet, wenn alle Items dieses Itemsets signifikant sind.

Generalisierung Ein Itemset $\hat{X}$ wird genau dann als *Generalisierung* eines Itemsets X bezeichnet, wenn $\hat{X}$ dieselben Attribute enthält wie X, und es gilt:

$$\forall A_i(l, r) \in X \wedge A_i(l', r') \in \hat{X} : l' \leq l \leq r \leq r'$$

D.h. alle Transaktionen, welche sich auf das Itemset X abbilden lassen, können ebenso auf das Itemset $\hat{X}$ abgebildet werden.

Im Folgenden soll ein kurzes Beispiel die eben beschriebenen Zusammenhänge verdeutlichen.

Beispiel 5.1 Gegeben sei ein Datenstrom, welcher Werte zweier Attribute A_1 (zum Beispiel Temperatur) und A_2 (zum Beispiel Bewegung) enthält. Im Zeitintervall Δt wurden die in Tabelle 5.2 dargestellten zehn Transaktionen beobachtet. Es seien weiterhin die folgenden Items gegeben:

$$A_1(20, 21.5], A_1(20, 23], A_2(0, 50] \text{ und } A_2(50, 100]$$

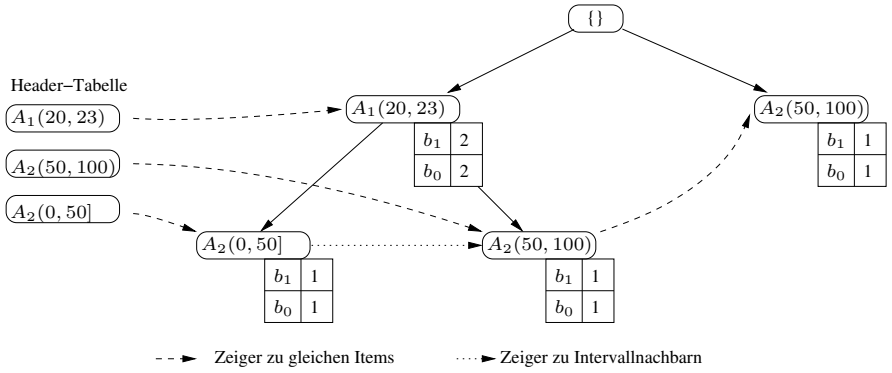


Abbildung 5.5: Beispiel FP^2-Stream

Basierend auf diesen Items lassen sich die in Tabelle 5.2 abgebildeten Transaktionen auf die in Tabelle 5.3 dargestellten Itemsets abbilden. Die Tabelle zeigt die entsprechenden Häufigkeiten und den Support, den die einzelnen Itemsets aufweisen.

Es lassen sich die folgenden Dichten für die Items ermitteln: $density(A_1(20, 21.5]) = 2$ und $density(A_1(20, 23]) \approx 2.7$. Bei dem einelementigen Itemset $\{A_1(20, 23]\}$ handelt es sich um eine Generalisierung des Itemsets $\{A_1(20, 21.5]\}$. ■

5.3.2 FP^2-Stream

Die meisten in der Literatur zu findenden Frequent-Pattern-Mining-Verfahren [18, 193], welche quantitative Attribute betrachten, zerlegen den Wertebereich für ein Attribut in äquidistante Intervalle, welche anschließend derart miteinander kombiniert werden, dass sie den geforderten Kriterien (minimaler Support und Signifikanz) genügen. Zu den wesentlichen Problemen dieser *Bottom-Up*-Strategie zählt dabei das Finden einer geeigneten Zerlegung [193] und das möglichst effiziente Generieren zusammenhängender Items aus den Intervallen.

Im Folgenden soll mit dem FP^2-Stream ein speichereffizientes Verfahren für quantitative Attribute vorgestellt werden, welches zudem für die Analyse über Datenströmen geeignet ist. Die Itemsets werden beim FP^2-Stream in einem Präfixbaum (ähnlich dem FP-Tree von Han et al. [101]) verwaltet. Das Ziel des vorgestellten Verfahrens ist die kontinuierliche Verfeinerung der Items (*Top-Down*-Strategie) und der daraus aufgebauten Itemsets beim Eintreffen neuer Transaktionen mit dem Ziel, dass diese den geforderten Kriterien genügen.

Zunächst soll die dem Algorithmus zugrundeliegende Datenstruktur erläutert werden. Anschließend folgt eine Beschreibung der Algorithmen zum Einfügen neuer Elemente beim Eintreffen von Transaktionen und zum Optimieren der Datenstruktur.

Input : Menge von Transaktionen D
Output : Menge von häufigen Itemsets
erzeuge leeren FP^2-Tree;
stelle für jedes Attribut A_i im Datenstrom dessen minimalen und maximalen Wert min und max fest und lege entsprechende Itemknoten im FP^2-Tree an;
füge die Transaktionen von Batch B_0 in den FP^2-Tree ein;
übernimm Knotengrenzen und Häufigkeiten aus dem FP^2-Tree in den FP^2-Stream;
while *Batch* B_i, $i > 0$ **do**
    initialisiere FP^2-Tree mit den Knoten und Intervallgrenzen des FP^2-Stream;
    sortiere alle Transaktionen des aktuellen Batch in den FP^2-Tree ein (falls notwendig, füge neue Knoten hinzu bzw. erweitere existierende Knoten);
    übertrage alle Knoten aus dem FP^2-Tree in den FP^2-Stream;
    führe eventuell notwendige Split-Operationen auf dem FP^2-Stream aus;
    führe eventuell notwendige Merge-Operationen auf dem FP^2-Stream durch;
    Lies alle häufigen Itemsets aus dem FP^2-Stream aus (FP^2-Growth);
return häufige Itemsets;

Algorithmus 5.5 : Prinzipieller Ablauf FP^2-Stream

5.3.2.1 Datenstruktur

Der FP^2-Stream entspricht einem Präfixbaum, in welchem die häufigsten Itemsets in einer kompakten Form gespeichert werden. Eine Header-Tabelle enthält Verweise auf alle Itemsets, welche sich gegenwärtig im Präfixbaum befinden. Jeder Pfad von der Wurzel bis zu einem Knoten im Präfixbaum repräsentiert ein Itemset. Zusätzlich zu den Itemsetinformationen sind in jedem Knoten des Baumes die Häufigkeiten der durch den Pfad des Knotens repräsentierten Itemsets in den letzten k Zeiträumen vermerkt (ähnlich dem FP-Stream von Gianella et al. [85]). Bei den Zeiträumen handelt es sich um gleitende Zeitfenster (statt der in [85] verwendeten logarithmischen Zeitfenster). Der Einsatz gleitender Zeitfenster führt zu einem „gezielten Vergessen“ alter Werte.

Weiterhin sind alle Knoten, welche das gleiche Item repräsentieren, untereinander über Zeiger verbunden. Der entsprechende Eintrag der Header-Tabelle verweist dabei auf das erste Element der so erzeugten Liste. Jeder Eintrag der Header-Tabelle enthält zusätzlich ein *Equi-Width*-Histogramm, welches einen approximativen Überblick über die Häufigkeitsverteilung der zuletzt eingefügten Werte in die Items gibt. Abbildung 5.5 zeigt einen Beispiel-FP^2-Stream, dem die in Tabelle 5.2 beschriebenen Transaktionen zugrunde liegen. Das Attribut A_2 wurde in der Abbildung bereits einer Verfeinerung unterzogen (siehe Abschnitt 5.3.2.3).

Im weiteren Verlauf dieses Abschnittes sollen die für die Verwaltung des Präfixbaumes notwendigen Operationen im Detail beschrieben werden. Die Verwaltung der einzelnen Items im FP^2-Stream folgt im wesentlichen einem *Top-Down*-Ansatz, d.h. die einzelnen Items (und somit die daraus gebildeten Itemsets) werden durch geeignete Split- und Merge-Operationen so lange verfeinert, bis sich Itemsets bilden, welche eine annähernd gleiche Informationsdichte aufweisen. Den generellen Ablauf des Verfahrens zeigt Algorithmus 5.5.

5.3.2.2 Einfügen neuer Transaktionen

Das Einfügen neuer Transaktionen in den FP^2-Stream erfolgt batchweise. Ein Batch B bezeichnet dabei eine Menge von $|B|$ aufeinanderfolgenden Transaktionen. Neue Transaktionen werden nicht direkt in den FP^2-Stream integriert, sondern zuvor in einen sogenannten FP^2-Tree, bei dem es sich ebenfalls um einen Präfixbaum handelt, eingefügt. Der FP^2-Tree entspricht dabei in wesentlichen Teilen dem FP^2-Stream, wobei die Knoten jedoch lediglich die zum aktuellen Batch gehörenden Häufigkeiten verwalten. Die Implementierung des FP^2-Tree erfolgt in Form von „Schattenfenstern“, welche in die Knoten des FP^2-Stream integriert werden. Somit fallen für das Anlegen des FP^2-Tree keine zusätzlichen Kosten an. Es muss lediglich beim ersten Einfügen einer Transaktion eines neuen Batches das entsprechende „Schattenfenster“ angelegt werden.

Ähnlich wie beim FP-Stream [85] wird das Einfügen des ersten Batches B_0 in den FP^2-Stream getrennt von der Behandlung aller weiteren Batches betrachtet. Zum Zeitpunkt des Einfügens von B_0 in den FP^2-Stream ist kein Wissen über die genaue Verteilung der Stromdaten vorhanden. Es stehen lediglich die Informationen aus dem ersten Batch zur Verfügung. Zunächst wird daher für jedes Attribut A_i des Datenstromes ein Item $A_i(min, max)$ angelegt, wobei min dem minimalen Wert von A_i in B_0 und max dem maximalen Wert von A_i in B_0 entspricht. Die Items werden anschließend, absteigend nach der Häufigkeit ihres Auftretens, im ersten Batch sortiert und in den FP^2-Tree eingefügt. Häufige Items werden also wurzelnah eingefügt. Die so bestimmte Einfüge-Reihenfolge der Attribute gilt im Anschluß für alle weiteren Batches. Da alle Items nach dem Einfügen des ersten Batches den kompletten aktuellen Wertebereich des Attributes überdecken, kann es ausschließlich durch das vollständige Fehlen von Attributwerten innerhalb von Transaktionen zu Unterschieden in den Häufigkeiten der einzelnen Items kommen.

Nachdem der erste Batch erfolgreich eingefügt wurde, werden alle weiteren Batches gleich behandelt. Jede Transaktion im Batch wird nach folgendem Schema zunächst in den FP^2-Tree eingefügt:

- Existiert bereits ein Knoten, welcher das Itemset repräsentiert, so wird die Frequenz des entsprechenden Knotens im FP^2-Tree um 1 inkrementiert.
- Existiert kein Knoten, welcher das Itemset repräsentiert, so muss ein neuer Knoten im FP^2-Tree angelegt werden. In diesem Fall gibt es zwei Möglichkeiten:
  - Der neu anzulegende Knoten wird sowohl auf der linken als auch auf der rechten Seite von existierenden Knoten eingeschlossen. Als Intervallgrenzen für den neuen Knoten werden die Grenzen der benachbarten Knoten gewählt. D.h., zwischen den beiden begrenzenden Knoten $A_i(l, r)$ und $A_i(l', r')$ wird ein neuer Knoten $A_i(r, l')$ angelegt.
  - Der einzufügende Wert über- bzw. unterschreitet die Intervallgrenzen aller existierender Knoten. Existiert ein Knoten im FP^2-Tree, welcher noch keinem Knoten im FP^2-Stream entspricht und dessen Intervallgrenzen sich

derart erweitern lassen, dass er den Wert aufnehmen kann, so werden die Grenzen dieses Knotens angepasst und dessen Frequenz entsprechend inkrementiert. Andernfalls muss ein neuer Knoten mit dem neuen Wert als Minimum bzw. Maximum als Grenze angelegt werden.

Wurden ausreichend Transaktionen in den FP^2-Tree eingefügt (die vorgegebene Batch-Größe wurde erreicht), dann kann dieser in den FP^2-Stream integriert werden. Hierzu wird jedem Knoten im FP^2-Stream ein Eintrag für den neuen Batch im gleitenden Zeitfenster hinzugefügt. Sollten im FP^2-Stream Knoten vorhanden sein, welche durch die Integration des FP^2-Tree keine Aktualisierung erfahren, so sind deren Häufigkeiten für diesen Batch 0. Sind während der Verarbeitung eines Batches neue Knoten im FP^2-Tree hinzugekommen, so müssen diese ebenfalls in den FP^2-Stream übernommen werden. Die Zeitfenster der neuen Knoten enthalten dabei nur den Häufigkeitswert des aktuellen Batches.

Wurden alle Transaktionen eines Batches in den FP^2-Stream eingefügt, dann wird anschließend geprüft, inwieweit sich Items verfeinern lassen. Ziel ist es dabei, eine möglichst hohe Informationsdichte pro Item zu erhalten, welche über das ganze Item gleichmäßig verteilt ist. Der Prozess der Verfeinerung ist hierbei ein zweistufiger Prozess. In einem ersten Schritt werden diejenigen Items verfeinert, deren Füllgrad zu hoch ist. Im zweiten werden Items, die sich generalisieren lassen, gemischt. Beide Prozesse werden im Weiteren genauer beschrieben.

5.3.2.3 Aufspaltung von Items

Als wesentliches Kriterium für die Güte der Approximation eines Items wird die Häufigkeitsverteilung der ihm zugehörigen Attributwerte im letzten Batch herangezogen. Sind die in ein Item eingefügten Werte ungleichmäßig verteilt, so wurden die Grenzen für dieses Item schlecht gewählt. Der Grad der Ungleichverteilung wird über die Schiefe der in der Header-Tabelle mitgeführten *Equi-Width*-Histogramme bestimmt. Es existiert dabei für jeden Batch und jedes Item ein eigenes Histogramm.

Die Schiefe als Maß für die Güte eines Items $A_i(l,r)$ wird wie folgt bestimmt:

$$skew(A_i(l,r)) = \frac{\max_{1\leq j\leq d}\{hist(A_i(l,r),j)\} - \min_{1\leq j\leq d}\{hist(A_i(l,r),j)\}}{\sum_{j=1}^{d} hist(A_i(l,r),j)} \tag{5.2}$$

Wobei d die Anzahl der Buckets im Histogramm bezeichnet. $hist(A_i(l,r),j)$ bezeichnet die Häufigkeit im j-ten Bucket des Histogramms für das Item $A_i(l,r)$.

Überschreitet die Schiefe für ein gegebenes Item $A_i(l,r)$ einen vorab definierten Schwellwert $maxskew$, d.h. $skew(A_i(l,r)) > maxskew$, dann wird das Item in zwei Items mit disjunkten Intervallen gesplittet. O.B.d.A. werden im Folgenden Items immer in Items mit links offenem Intervall zerlegt. Für den Teilungspunkt des ursprünglichen Intervalls werden im Folgenden zwei Strategien vorgestellt:

1. Das Intervall des aufzuspaltenden Items wird halbiert.

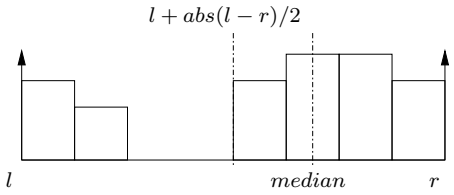


Abbildung 5.6: Intervallteilung beim Aufspalten eines Items

2. Der Median über die in dem letzten Batch eingefügten Werte wird für die Ermittlung des Teilungspunktes herangezogen.

Im Fall der Intervall-Halbierung werden aus Item $A_i(l, r)$ die beiden disjunkten Items $A_i(l, k]$ und $A_i(k, r)$ mit $k = l + abs(l - r)/2$ erzeugt.

Ziel des medianbasierten Ansatzes ist die Berücksichtigung der realen Häufigkeitsverhältnisse innerhalb eines Items. Hierzu wird der Median über die Buckets des Häufigkeitshistogramms bestimmt. Die Aufspaltung des Itemintervalls erfolgt anschließend auf Basis des Mittelpunktes des Median-Buckets. Das Ausgangsitem $A_i(l, r)$ wird dabei in die beiden Items $A_i(l, median/2]$ und $A_i(median/2, r)$ zerlegt. Abbildung 5.6 zeigt die beiden Verfahren zur Intervallhalbierung im Vergleich.

Auch der Median-Ansatz berücksichtigt lediglich die Gesamtverteilung aller Knoten, die ein bestimmtes Item repräsentieren. Existieren für ein Item 2 Knoten im FP^2-Tree, bei denen sich die Buckethäufigkeiten genau entgegengesetzt verhalten, dann ergibt dies für das Gesamthistogramm des Items eine gleichmäßige Häufigkeitsverteilung. Dies spiegelt nicht die tatsächliche Verteilung wider.

Bei einer Itemaufspaltung müssen alle Knoten im Baum, welche dieses Item repräsentieren, ebenfalls aufgespalten werden. Befinden sich unterhalb eines Knotens, welcher ein Item repräsentiert, das aufgespalten wird, weitere Knoten, dann werden diese Teilbäume kopiert und als neue Teilbäume an die entstehenden Knoten angehangen. Die Häufigkeit des mit den aufgespalteten Knoten assoziierten Itemsets wird beim Überführen in die neuen Itemsets halbiert. Die genaue Verteilung der Daten in den einzelnen Knoten ist unbekannt, weswegen der Einfachheit halber eine Gleichverteilung angenommen wird. Ein Beispiel soll die Itemaufspaltung verdeutlichen.

Beispiel 5.2 Es wird angenommen, dass das Item $A_1(20, 23]$ aus dem vorigen Beispiel den definierten Schwellwert *maxskew* überchreitet. Bei der Itemaufteilung auf Basis der Intervall-Halbierung wird das Item in die beiden Subitems $A_1(20, 21.5]$ und $A_1(21.5, 23]$ geteilt. Es entstehen somit die abhängigen 2-Itemsets

$$\{A_1(20, 21.5], A_2(0, 50]\},\ \{A_1(20, 21.5], A_2(50, 100]\},$$
$$\{A_1(21.5, 23], A_2(0, 50]\},\ \{A_1(21.5, 23], A_2(50, 100]\},$$

welche jeweils eine geschätzte Häufigkeit von 1 besitzen. Abbildung 5.7 zeigt den entsprechenden FP^2-Stream nach dem Aufspalten des Items. ■

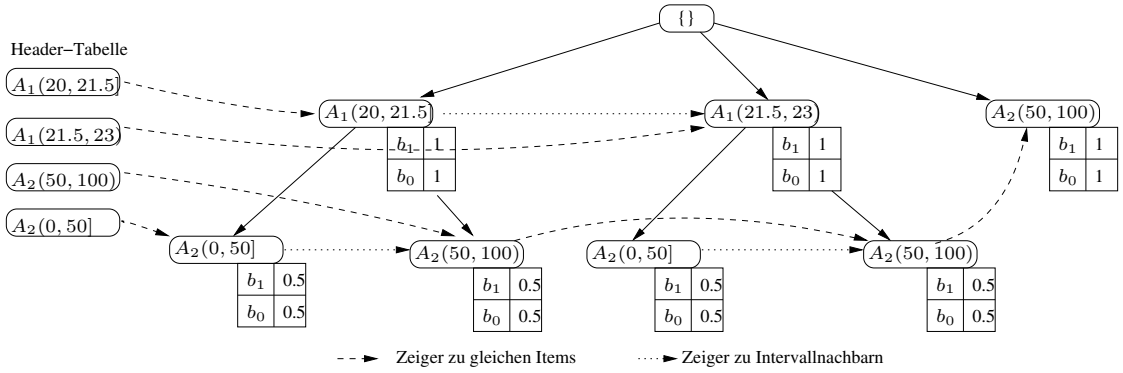


Abbildung 5.7: Aufspalten von Knoten $A_1(20, 23]$

Minimale Intervallgröße Eine beliebige Verfeinerung der Intervalle ist aus Gründen der effizienten Datenhaltung nicht sinnvoll. Somit ergibt sich die Fragestellung, inwieweit Intervalle verfeinert werden sollten. In [193] untersuchen die Autoren, in wieviele Partitionen das durch den Wertebereich des Attributes gegebene Intervall maximal zerlegt werden muss, um einen möglichst geringen Informationsverlust durch die Generalisierung von Items zu garantieren. Als Maß für den Informationsverlust führen sie die *partielle Vollständigkeit* ein. Als K-partiell-vollständig wird dabei der maximale Support bezeichnet, den ein Itemset aufweisen darf (als K-Vielfaches des minimalen Supports). Basierend auf der partiellen Vollständigkeit zeigen die Autoren, dass im Fall einer Partitionierung des Attributwertebereiches in Basisintervalle gleicher Größe maximal $\frac{2 \cdot \alpha}{minsup \cdot (K-1)}$ Intervalle notwendig sind. α bezeichnet dabei die Anzahl quantitativer Attribute im Datenstrom. Daraus lässt sich die minimale Intervallgröße für jedes Datenstromattribut A_i ableiten. Zum Zeitpunkt des ersten Erstellens des FP^2-Stream (Batch B_0) wird für jedes Attribut A_i dessen kleinster bisher aufgetretener Wert min und dessen größter bisher aufgetretener Wert max bestimmt. Damit ergibt sich die minimale Intervallgröße $minIntSize$ zu

$$minIntSize = \frac{|max - min|}{\frac{2 \cdot \alpha}{minsup \cdot (K-1)}} \tag{5.3}$$

Beim späteren Eintreffen von Transaktionen mit Werten, welche das rechts- bzw. linkseitige Extremum erweitern, muss das minimale Intervall entsprechend angepasst werden.

Ringlisten Zusätzlich zu den Knoten müssen auch die der Batches entsprechenden Fenster angepasst werden. Jeder der beiden durch die Aufspaltung neu entstandenen Knoten erhält hierzu die Hälfte der Häufigkeitswerte des Original-Items. Um später bei der Itemsetextkraktion auf einfache Weise das bisherige Aufspaltungsverhalten von Items nachvollziehen zu können, wird bei einer Aufspaltung in jedem Zeitfensterslot ein Verweis auf das Fenster angelegt, das beim Aufspalten abgeteilt wurde. Um weitere Aufspaltungen bzw. das Zusammenführen möglichst effizient gestalten zu können, verweist das letzte Fenster in der Liste auf das erste Listenelement, so dass ein Ring

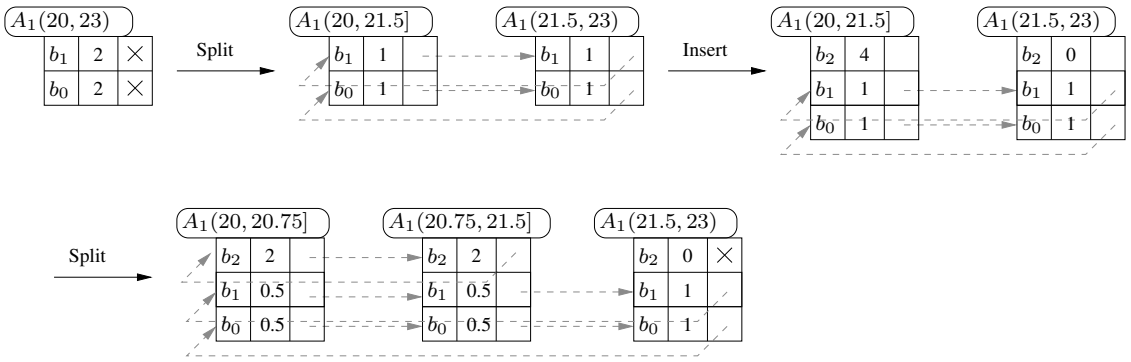


Abbildung 5.8: Fenster bei einem Itemsplit

entsteht. Der Ring ermöglicht anschließend eine schnelle Suche auch ohne zum Beispiel eine doppelt verkettete Liste. Abbildung 5.8 zeigt das wiederholte Teilen eines Items am Beispiel des Items $A_1(20, 23]$.

Mit jedem neu eingefügten Batch wird ein alter Batch aus den Fenstern verdrängt, so dass der damit verbundene Ring ebenfalls entfernt wird. Das kontinuierliche Verdrängen alter Werte in den Fenstern hat zur Folge, dass sich die exakten Häufigkeitswerte in den verfeinerten Intervallen über die Zeit durchsetzen.

5.3.2.4 Pruning und Reorganisation

Durch das Aufspalten von Items wird der FP^2-Stream kontinuierlich vergrößert. So resultiert zum Beispiel der Split des Wurzelknotens in einer Verdoppelung aller Knoten in der Datenstruktur. Um diese dennoch möglichst klein und dessen Verarbeitung effizient zu gestalten, werden verschiedene Ansätze verfolgt:

- Itemsets, welche gegenwärtig nicht mehr häufig sind und in den nächsten Zeitschritten auch nicht mehr häufig werden können, werden aus der Datenstruktur entfernt.
- Im FP^2-Stream bestimmt der erste Batch die Einfügereihenfolge von Items (ähnlich dem FP-Stream). Da sich die Datenverteilung und damit die Häufigkeiten der einzelnen Items über die Zeit ändern können, kann eine Reorganisation der Datenstruktur zu einer deutlichen Verkleinerung führen.

Pruning Das Entfernen nicht notwendiger Itemsets und Teilfenster erfolgt im Original-FP-Stream während der Pruning-Phase (genauer dem *tail pruning*). Während dieser Phase wird geprüft, ob (i) Knoten leer sind und entfernt werden können und (ii) ob Teilfenster eventuell entfernt werden können. Der hier beschriebene Ansatz des FP^2-Stream verwendet gleitende Fenster, was ein kontinuierliches tail pruning zur Folge hat. Ein zusätzliches Pruning, wie zum Beispiel in [85] beschrieben, wird gegenwärtig nicht durchgeführt.

Reorganisation Eines der wesentlichen Probleme des FP^2-Stream bzw. des FP-Stream ist die Abhängigkeit von den Häufigkeitswerten des ersten Batches. Die richtige Wahl hat einen entscheidenden Einfluss auf die Struktur des Präfixbaumes und damit auf die Performance des Verfahrens. Dieses Problem tritt insbesondere beim FP^2-Stream zu Tage. Jede Knotenverfeinerung hat die Verdoppelung der Anzahl seiner Kindknoten zur Folge. Wurde die initiale Struktur schlecht gewählt, so resultiert dies im schlimmsten Fall in einem exponentiellen Wachstum der Knotenanzahl (bis zum Erreichen der minimalen Intervallgröße). Entsprechend steigen Speicher und Wartungsaufwand der Datenstruktur. Eine kontinuierliche Überwachung und Re-Optimierung ist somit unumgänglich.

Ein einfacher Ansatz zur Optimierung der Datenstruktur ist die Extraktion aller Itemsets aus dem aktuellen FP^2-Stream, die Ordnung dieser nach ihren Häufigkeiten und das anschließende Neubefüllen eines leeren FP^2-Stream. Die effiziente Reoptimierung des FP^2-Stream ohne das vollständige Neubauen dieser ist ein offenes Problem. In [199,200] wurde eine Restrukturierungsphase beschrieben, welche das gleiche Problem für einen anderen Präfixbaum löst.

5.3.2.5 Item-Verschmelzung

Eine weitere Möglichkeit, den Präfixbaum kompakt zu halten, ist das Verbinden benachbarter Knoten mit nahezu gleicher Informationsdichte. Diese können nahezu ohne Informationsverlust kombiniert werden. Im Weiteren soll der Knotenverbund im Detail beschrieben werden.

Gegeben seien die beiden Items $A_i(l, r]$ und $A_i(l', r']$. Diese können genau dann zu einem neuen Item $A_i(l, r']$ zusammengefasst werden, wenn für alle Knoten im FP^2-Stream, die dieses Item repräsentieren, die folgenden Bedingungen erfüllt sind:

- Die Items $A_i(l, r]$ und $A_i(l', r']$ sind *direkte Nachbarn*, d.h. es gilt $r = l'$.
- Die Items $A_i(l, r]$ und $A_i(l', r']$ besitzen den gleichen Präfix, d.h. sie verfügen über den gleichen Elternknoten im FP^2-Stream.
- Die Informationsdichte des Items $A_i(l, r]$ entspricht nahezu der Informationsdichte des $A_i(l', r']$, d.h. $density(A_i(l, r]) \sim density(A_i(l', r'])$.

Die erste Bedingung stellt sicher, dass es sich bei den Verbundkandidaten auch wirklich um Intervallnachbarn handelt. Ausschließlich direkte Nachbarn können miteinander verbunden werden. Die zweite Bedingung garantiert, dass alle von der Item-Verschmelzung betroffenen Itemsets ebenfalls verbunden werden können. Die letzte Bedingung stellt sicher, dass die zu verbindenden Items über einen nahezu gleichen Informationsgehalt verfügen. Es kommt damit beim Verschmelzen zu keinem Informationsverlust bzw. der Informationsverlust bewegt sich in einem vom Nutzer definierten Bereich.

In Algorithmus 5.5 wird die mögliche Verschmelzung umgehend nach dem Split ausgeführt. Wurde ein Split auf einem Item durchgeführt, dann verfügen die beiden resultierenden Items über die gleiche Dichte. Erst durch das Einfügen neuer Batches setzt sich die Schiefe, welche zuvor als Voraussetzung für das Aufteilen notwendigerweise festgestellt wurde, auch in den neuen Items durch. Aus diesem Grund ist die sofortige Rekombination zuvor erst geteilter Items für die nächsten i Batches nicht möglich. Auf der anderen Seite sollte die Verzögerung um i Batches kleiner gewählt werden als die Anzahl der Batches in den Zeitfenstern der Knoten. Damit wird verhindert, dass sich kurzfristige Schwankungen in der Dichte auch in der Extraktion der häufigen Itemsets durch den FP^2-Growth durchsetzen.

5.3.2.6 Item-Rekonstruktion und Itemset-Extraktion

Zwar werden im FP^2-Stream häufige Itemsets in einer kompakten Darstellung gespeichert, es wird aber dadurch nicht garantiert, dass diese auch effizient extrahiert werden können. Mit der Entwicklung des FP-Tree präsentierten Han et al. in [101] FP-Growth ein Verfahren zur effizienten Extraktion aller im FP-Tree gespeicherten häufigen Muster. Das Herauslösen der häufigen Itemsets stellt dabei ein kombinatorisches Problem dar. Im Verlauf der letzten Jahre wurden neben dem von Han et al. präsentierten Ansatz weitere Verfahren vorgestellt, welche sich im Wesentlichen in die Klassen *Top-Down*- [212] bzw. *Bottom-Up*-Verfahren [37, 169] einordnen lassen. Ein großer Vorteil von Top-Down-Verfahren ist, dass die Konstruktion von *conditional pattern-bases* und damit die aufwendige Konstruktion von Teilbäumen verhindert werden kann.

Im Folgenden wird ein für den FP^2-Stream angepasstes Growth-Verfahren zum Extrahieren der häufigen Muster beschrieben. Das beschriebene Verfahren bestimmt dabei zum Zeitpunkt des Ausführens immer die vollständige Itemset-Menge zum Ausführungszeitpunkt. Oftmals ist für den Anwender allerdings die Information, dass sich etwas geändert hat, ausreichend. Daher soll anschließend ein einfacher Ansatz zur Identifikation von Änderungen über Itemsets unter Verwendung des FP^2-Stream gezeigt werden.

FP^2-Growth Bevor die häufigen Itemsets aus dem FP^2-Stream extrahiert werden können, sind zwei Vorverarbeitungsschritte durchzuführen. In einem ersten Schritt wird aus dem FP^2-Stream ein FP^2-Tree extrahiert. Die Häufigkeiten der Knoten im FP^2-Tree ergeben sich dabei aus der Summe über alle Fenster, die den entsprechenden Knoten im FP^2-Stream repräsentieren. Aufgrund des kontinuierlichen Teilens von Knoten und dem damit verbundenen Aufteilen von Häufigkeitswerten (unter Annahme einer Gleichverteilung der einzelnen Werte in den Intervallen) handelt es sich bei den Häufigkeiten in den Knoten des FP^2-Stream zumeist nur um approximierte Werte. Lediglich Knoten, welche zuvor nicht geteilt wurden, weisen genaue Häufigkeitswerte auf. In der gegenwärtigen Umsetzung wird gefordert, dass Items nur auf Basis der tatsächlichen Häufigkeiten korrekt erfasst werden. Aus diesem Grund werden in den FP^2-Tree nur Knoten mit genauen Häufigkeitswerten integriert. Aufgeteilte Knoten

mit approximierten Häufigkeiten werden somit zusammengefasst, bis ein Knoten entsteht, welcher genaue Häufigkeiten enthält. Die Ermittlung der exakten Häufigkeiten erfolgt effizient unter Verwendung der Ringlisten über den Zeitfenstern (die Summe über alle Knoten einer Ringliste). Prinzipiell ist es auch denkbar, Itemsets über approximierten Häufigkeitswerten zu erstellen. Dies erfordert entsprechende Einschätzungen bezüglich der Ergebnisgüte und soll an dieser Stelle nicht weiter betrachtet werden.

Der durch den Extraktionsprozess erzeugte FP^2-Tree enthält im Allgemeinen quantitative Items, deren Support nicht dem geforderten minimalen Support *minsup* genügt. In einem nächsten Schritt werden daher durch Rekombination benachbarter Intervalle Items erzeugt, welche dem minimalen Support genügen. Die maximal zu erzeugende Anzahl α (siehe Abschnitt 5.3.2.3) an Items pro Datenstromattribut A_i ist dabei vorab zu definieren (entsprechend [193]). Während der Rekombinationsphase wird das Item mit der höchsten Informationsdichte so lange mit benachbarten Items (dem jeweils direkt benachbarten Item mit der höchsten Informationsdichte) verbunden, bis der geforderte minimale Support erreicht wird. Soll mehr als ein häufiges Item pro Datenstromattribut ermittelt werden, so wird das beschriebene Verfahren auf die verbliebenen nicht-häufigen Items angewandt.

Nachdem der FP^2-Tree erzeugt wurde, dessen Items den definierten Bedingungen genügen, können aus diesem häufige Itemsets mit den oben erwähnten Verfahren extrahiert werden. Aus Gründen der effizienten Auswertung wird hierzu auf das *Top-Down*-Verfahren nach [212] zurückgegriffen.

Änderungserkennung mittels FP^2-Growth Oftmals ist es für den Anwender von größerem Interesse zu erfahren, ob sich die Menge der häufigen Itemsets mit der Zeit verändert, als bei jeder Änderung eine vollständige Liste der häufigen Itemstes zu erhalten. Prinzipiell kann ein solches Ergebnis erreicht werden, indem das zuletzt erhaltene Ergebnis mit den aktuellen Resultaten verglichen wird. Probleme wie zum Beispiel die Umordnung der Ergebnismenge, das Erkennen von fehlenden Itemsets und das Hinzufügen neuer Itemsets erschweren diese Aufgabe (ähnlich dem *differential snapshot problem* [137]). Aus diesem Grund scheint es geeignet zu sein, den FP^2-Growth direkt um die Möglichkeit der Erkennung von Änderungen gegenüber dem letzten Test zu erweitern, anstatt die Änderungserkennung im Anschluss an das Verfahren durchzuführen.

Für die Änderungserkennung im Verfahren selbst müssen zunächst die Häufigkeiten der letzten Analyse durch den FP^2-Growth effizient gespeichert werden. Zu diesem Zweck wird in den FP^2-Stream zusätzlich ein FP^2-Tree eingebettet, welcher die Häufigkeitswerte des letzten Durchlaufes speichert (wiederum in Form von „Schattenknoten“). Bei der nächsten Aktivierung des FP^2-Growth werden diese Häufigkeitswerte für die Differenzenbestimmung herangezogen. Ein Itemset wird genau dann der Ausgabemenge hinzugefügt, wenn eine der nachstehenden Bedingungen zutrifft:

- Im letzten Durchlauf war das Itemset noch nicht häufig, in diesem Durchlauf überschreitet es den geforderten minimalen Support.

- Im letzten Durchlauf war das Itemset häufig, in diesem Durchlauf ist es nicht mehr häufig.
- Das Itemset war bereits häufig und ist dies auch weiterhin. Die Intervallgrenze mindestens eines Items des Itemsets hat sich verändert.

War ein Knoten im letzten Iterationsschritt noch nicht im FP^2-Stream enthalten, so wird für diesen eine Häufigkeit von 0 angenommen. Gleiches gilt für den Fall, dass ein Knoten aus dem FP^2-Stream entfernt wurde. Veränderungen von Itemsets, welche zu keinem Zeitpunkt häufig waren, werden der Ergebnismenge in keinem Fall hinzugefügt (analog dem Standard FP^2-Growth).

5.3.3 Integration in AnduIN

In einem ersten Schritt wurde das oben beschriebene Verfahren als Operator für die DSMS-Komponente von *AnduIN* implementiert. Im Weiteren wurden verschiedene Operatorimplementierungen geschaffen, welche teilweise im WSN bearbeitet werden können und ihre Zwischenergebnisse an die zentrale Komponente senden [178]. Diese führt dann die abschließende Verarbeitung des Operators aus. Im weiteren Verlauf dieses Abschnittes werden die verschiedenen verteilten Lösungen kurz präsentiert. Anschließend erfolgt eine Beschreibung der Integration in das in Abschnitt 4.2 vorgestellte Operatormodell.

5.3.3.1 Partielle In-Network-Verarbeitung

Die Reorganisation der Datenstruktur durch Aufspalten und Zusammenfügen von Items gehört neben dem Auslesen der häufigen Itemsets zu den komplexesten Operationen im FP^2-Stream. Beim Aufspalten und Zusammenfügen müssen alle Items geprüft und u.U. sämtliche Knoten im Baum analysiert werden. Da im Fall einer verteilten Implementierung hierzu regelmäßig Nachrichten zwischen den einzelnen Sensorknoten ausgetauscht werden müssten, wurden diese Operationen als Teil der zentralen Verarbeitung des FP^2-Stream Operators umgesetzt. Da eine Weiterverarbeitung der aus der Datenstruktur extrahierten häufigen Muster lediglich auf der zentralen Komponente sinnvoll erscheint, wurde der FP^2-Growth ebenfalls auf dieser belassen.

Beim FP^2-Stream sind in den einzelnen Knoten gleitende Fenster eingebettet. Da diese ausschließlich vom FP^2-Growth-Verfahren ausgewertet werden, ist eine Auslagerung dieser in das WSN ebenfalls nicht sinnvoll. Letztendlich erscheint lediglich die In-Network-Verarbeitung des letzten Batches (einem FP^2-Tree) vielversprechend. Im Weiteren sollen hierfür zwei Ansätze vorgestellt werden.

Batchweise Verarbeitung Die Auswertung der im FP^2-Stream gespeicherten Häufigkeiten erfolgt immer erst am Ende eines Batches. Während der Einfügephase wird die

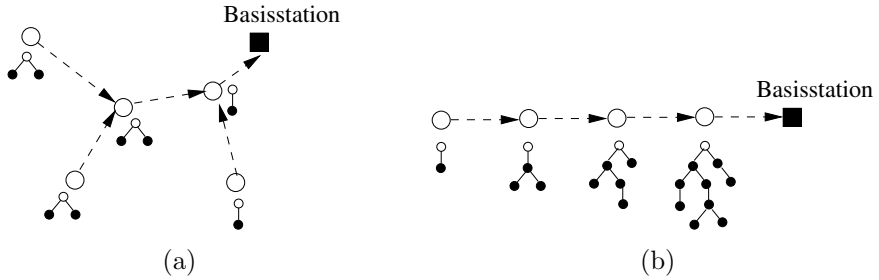


Abbildung 5.9: Verteilte Verarbeitung des FP^2-Stream: (a) batchweise Verarbeitung und (b) Präfixbaum-batchweise Verarbeitung

Synopse lediglich mit den neu eintreffenden Werten befüllt. Die Daten eines einzelnen Batches müssen erst zum Ende des Batches auf der zentralen Instanz zur Verfügung stehen. Das nachstehende Verfahren macht von dieser Eigenschaft Gebrauch.

Jeder Sensorknoten verwaltet den Teil des FP^2-Tree, den er selbst mit Daten befüllt. M.a.W. auf einem Sensorknoten liegen genau diejenigen Knoten, welche von diesem mit Werten versorgt werden. Zusätzlich liegen noch die Histogramme der Header-Tabelle der betroffenen Items verteilt im Netzwerk vor. Ein Histogramm befindet sich genau dann auf einem Sensorknoten, wenn dieser die zum Histogramm gehörenden Items verwaltet.

Damit der zeitliche Zusammenhang zwischen den Messwerten der Sensoren nicht verloren geht, müssen die Messwerte am Ende eines Batches an die zentrale Instanz weitergeleitet werden. Dadurch, dass die Sensoren wissen, welche Items es im FP^2-Stream gibt, können sie die Messwerte in Form der von ihnen verwalteten Knoten versenden. Durch das Auffüllen der Nachrichten mit Daten und die Zuordnung in die Knoten ist der Kommunikationsaufwand geringer, als wenn die Messwerte direkt versendet werden müssten. Zusätzlich zu diesen Informationen werden am Ende eines Batches die in den Sensorknoten verwalteten Histogramme ausgewertet. Lediglich die Schiefe der Histogramme sowie der Median werden an die zentrale Instanz übermittelt (siehe Abbildung 5.9(a)).

Die zentrale Instanz fügt die aus dem WSN erhaltenen Batches in den zentral verwalteten FP^2-Stream ein. Anschließend folgen die Schritte der Item-Verfeinerung und der Itemsetextraktion. Eventuelle Veränderungen an den Itemgrenzen müssen anschließend durch die zentrale Instanz an die Sensorknoten propagiert werden.

Präfixbaum-batchweise Verarbeitung Im eben beschriebenen Ansatz sind für das Update der zentralen Datenstruktur am Ende eines jeden Batches mindestens $h \cdot n$ Nachrichten notwendig (h bezeichnet die mittlere Hop-Entfernung der Knoten von der Basisstation und n die Anzahl der Knoten im WSN). Der folgende Ansatz, welcher eine zusätzliche Aggregation der Item-Information entlang einer Kettenstruktur, ähnlich dem *chained-based aggregation* Ansatz in [144], vornimmt, soll diesem Umstand entgegen wirken.

Bei diesem Ansatz verwaltet jeder Knoten im WSN einen Teilbaum des FP^2-Tree. Der von der Basisstation am weitesten entfernte Knoten verwaltet die Wurzel und alle von ihm bedienten Items des FP^2-Tree. Der nächste Knoten in Richtung Basisstation verwaltet eine vollständige Kopie der Datenstruktur seines Vorgängers in der Kette. Zusätzlich beinhaltet der von ihm gespeicherte Teil-FP^2-Tree noch alle Knoten, welche durch das Hinzufügen seiner eigenen Daten erzeugt werden können. Dieser sukzessive Aufbau der Datenstruktur setzt sich in Richtung Basisstation fort. Der letzte Knoten in der Kette, welcher zum Ergebnis beiträgt, verwaltet somit eine vollständige Kopie des FP^2-Tree.

Am Ende eines Batches werden die Änderungen beginnend beim letzten Element der Kette in Richtung Basisstation propagiert. Die Updates werden dabei von einem zum anderen Knoten in der Kette weitergereicht, wobei jeder Knoten zunächst mit Hilfe der Zwischenergebnisse seine lokale Kopie des FP^2-Tree anpasst. Anschließend werden die Änderungen des Knotens und seiner Vorgänger in aggregierter Form an den Nachfolgeknoten weitergereicht. Dieser passt die Datenstruktur seinerseits an usw. (Abbildung 5.9(b)). Die Auswertung des vollständigen Baumes und dessen Anpassungen werden am Batchende auf der zentralen Instanz vorgenommen.

Beim eben vorgestellten Ansatz werden zwischen zwei Sensorknoten im Mittel mehr Daten ausgetauscht als im vorhergegangenen Fall. Diese werden aber lediglich zum direkten Nachfolger gesendet. Somit entfällt das aufwendige Routing zur zentralen Instanz. In Abhängigkeit von der zugrundeliegenden Topologie (für die batchweise Verarbeitung) und den FP^2-Tree-Eigenschaften (Anzahl Attribute und Knoten je Attribut) sollte dieser Ansatz energetisch günstiger sein als die zuvor vorgestellte Lösung.

5.3.3.2 Umsetzung im Operatormodell

Da bei beiden Verarbeitungsvarianten ein Teil der Arbeit auf der zentralen Komponente stattfindet, müssen die Operatoren für *AnduIN* entsprechend zerlegt werden (siehe Abschnitt 4.1). Da die auf den Sensorknoten ausgeführten Teile der Operatoren von der zentralen Instanz über Änderungen an den von ihnen verwalteten Knoten und Items informiert werden müssen, sollten diese über eine entsprechende Rückkopplung verfügen. Bei dem vorgestellten Verfahren der einfachen batchweisen Verarbeitung erfolgt die Kommunikation der Knoten direkt mit der Operatorkomponente. Nach der Klassifikation in Abschnitt 4.1 handelt es sich bei dieser FP^2-Stream-Umsetzung um einen feedback-basierten Operator mit teilweiser Verarbeitung im WSN.

Bei der Präfixbaum-batchweisen Verarbeitung kommt zusätzlich die Kommunikation zwischen den Sensorknoten hinzu, da ein Knoten der Kette seine Zwischenergebnisse an seinen Nachbarknoten schickt. Trotzdem handelt es sich auch bei diesem Operator um einen feedback-basierten Operator mit teilweiser Verarbeitung im WSN. Der letzte Sensorknoten der In-Network-Verarbeitungskette sendet seine Zwischenergebnisse an die zentrale Komponente des Operators, welche dann die abschließenden Verarbeitungsschritte vornimmt.

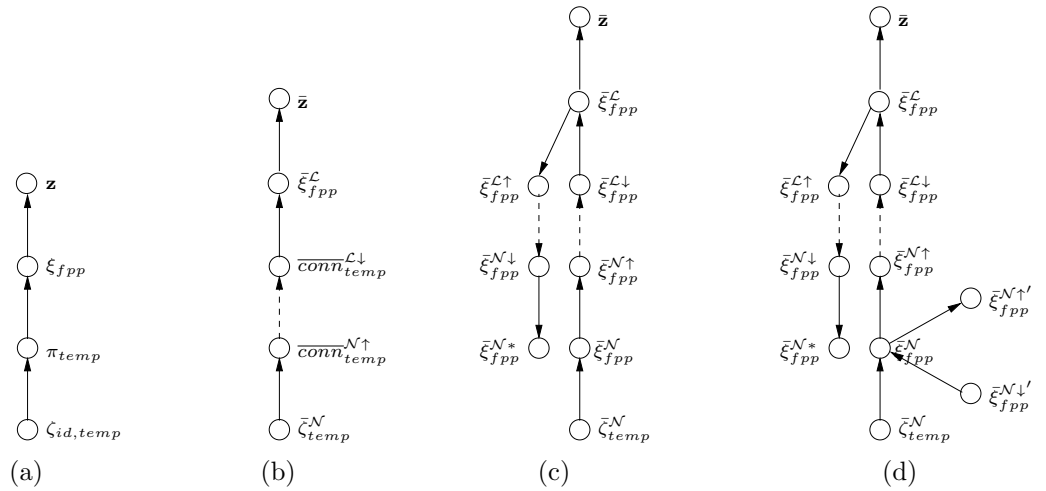


Abbildung 5.10: Beispiel 5.3: (a) logischer Anfrageplan, (b)-(d) mögliche physische Ausführungspläne

Ein einfaches Beispiel soll die Möglichkeiten der physischen Optimierung von *AnduIN* unter Verwendung der hier vorgestellten Verfahren zeigen.

Beispiel 5.3 Gegeben sei eine Sensornetzwerkquelle *net_stream*, welche die beiden Attribute *id* und *temp* zur Verfügung stellt. Darauf basierend wird nachstehende CQL-Anfrage definiert:

```
SELECT temp FROM net_stream [fppstream ()]
```

Basierend auf dieser Anfrage wird der in Abbildung 5.10(a) dargestellte logische Anfrageplan erzeugt. Die Abbildungen 5.10(b) bis 5.10(d) zeigen die drei physischen Ausprägungen des Beispiels. In Abbildung 5.10(b) ist der einfachste Fall, eine vollständige zentrale Verarbeitung, zu sehen. Die Projektion im logischen Anfrageplan findet sich im physischen Ausführungsplan im angepassten Schema des Sampling-Operators wieder. Abbildung 5.10(c) zeigt die Umsetzung des physischen Ausführungsplans für den Fall der batchweisen Verarbeitung. Der komplexe Operator wurde dabei, wie in Abschnitt 4.1 beschrieben, in eine Menge von einzelnen Operatoren zerlegt. Ähnlich gestaltet sich das Operatormodell des Präfixbaum-Ansatzes in Abbildung 5.10(d). Hier sind zusätzliche Operatoren für die Inter-Knoten-Kommunikation eingefügt worden. ■

5.3.3.3 Skalierbarkeit des Ansatzes

Sowohl mit steigender Attributanzahl als auch im Zuge der Verfeinerung von Items vergrößert sich die Datenstruktur. Da die Attributanzahl mit der Menge der Sensorknoten im WSN korreliert und die Speicherressourcen der einzelnen Sensorknoten stark beschränkt sind, ist eine Übertragung der Verfahren auf ein beliebig großes WSN prinzipiell nicht ohne weiteres möglich. Eine mögliche Strategie zur Lösung des Problems ist

die Partitionierung des WSN. Die Analyse findet dann anschließend jeweils nur über einem Teil des WSN statt. Prinzipiell können beliebig viele Partitionen existieren, welche parallel analysiert werden, solange die Größe jeder einzelnen Partition für die Analyse hinreichend klein ist.

Der von den Autoren in [179] vorgestellte Ansatz basiert auf diesem Prinzip. Auf Grundlage vorab definierter Eigenschaften werden im WSN Nachbarschaften erzeugt. Die Daten so definierter benachbarter Sensorknoten werden anschließend gemeinsam analysiert. Da der vorgestellte Ansatz auf einem angepassten Apriori-Algorithmus über kategorischen Werten basiert, kann er für unser Problem nicht verwendet werden.

In der vorliegenden Arbeit wurde ein anderer, vollständig manueller Ansatz umgesetzt. Der Nutzer definiert hierbei mittels der in Abschnitt 3.3.1.1 beschriebenen Syntax ein logisches Overlay für die Bearbeitung des FP^2-Stream. Im Falle des zweiten vorgestellten Verfahrens ist dies eine einfache Kette. Das eingesetzte Verfahren verwendet die so spezifizierte Struktur für das Routing der Knoten untereinander.

5.3.4 Verwandte Arbeiten

Eines der ersten Verfahren zum Finden von häufigen Mustern war das von Aggrawal et al. präsentierte *Apriori*-Verfahren [9, 10]. Die Grundlage des Ansatzes bildet die Annahme, dass sich häufige Muster ausschließlich aus häufigen Teilmustern zusammensetzen. Basierend auf dieser Heuristik erzeugt das Verfahren alle möglichen $k-Itemset$-Kandidaten aus der Menge aller $(k-1)-Itemset$-Kandidaten. Aufgrund der wiederholten Analyse der Daten handelt es sich hierbei um ein *Multi-Pass*-Verfahren, welches seine Anwendung insbesondere in der Verarbeitung von endlichen Datenmengen findet.

Auf Basis des Apriori-Algorithmus wurde eine Vielzahl weiterer Algorithmen entwickelt, welche unter anderem eine Steigerung der Performance oder die Adaption des Algorithmus an Datenströme zum Ziel hatten. Die Autoren in [172] präsentierten zum Beispiel mit dem DHP (*Direct Hashing and Pruning*) ein Verfahren, dessen Ziel die Optimierung der Anzahl an Kandidaten ist. Das Verfahren setzt dabei auf eine Hash-Tabelle zur Verwaltung von Kandidaten, deren Support den minimal geforderten Wert überschreitet.

Han et al. präsentieren in [101] mit dem FP-Tree (*frequent pattern tree*) einen Präfixbaum zur effizienten Speicherung häufiger Muster. Zusätzlich zu dieser Datenstruktur stellen sie mit dem FP-Growth-Algorithmus ein Verfahren zum Finden häufiger Muster auf Basis des FP-Tree vor. Im Gegensatz zum Apriori-Verfahren verzichtet das FP-Growth-Verfahren vollständig auf die Generierung von Kandidaten. Auch sind lediglich zwei Durchläufe durch die zu prüfende Datenmenge notwendig.

Frequent Pattern Mining über Datenströmen stellt eine besondere Herausforderung dar. Neben dem von Gianella in [85] beschriebenen FP-Stream existiert auch eine Vielzahl weiterer Lösungen. Der in [199] vorgestellte Compact Pattern Tree (CPT) zum Beispiel verwendet gleitende Fenster (ähnlich dem hier vorgestellem FP^2-Stream und dem DSTree [140]). Die Effizienz des Verfahrens wird dabei durch eine zusätzliche

Restrukturierungsphase erreicht. Während dieser wird der CPT in Abhängigkeit der Itemset-Häufigkeiten mit dem Ziel einer hohen Kompaktheit umsortiert. In [152] beschreiben die Autoren neben dem Sticky-Sampling und dem Lossy-Counting mit BTS (*Buffer-Trie-SetGen*) ein Verfahren, welches die Häufigkeiten von Itemsets mittels einer Gitter-Struktur speichert.

Die erste Arbeit, die sich mit dem Problem des quantitativen Frequent-Itemset-Mining beschäftigte, stammt von Srikant et al. [193]. Das vorgestellte Verfahren basiert dabei im Wesentlichen auf einem Apriori-Verfahren, welches die partitionierten Wertebereiche der einzelnen Attribute in geeigneter Weise kombiniert. Die Autoren in [76] beschreiben ein Verfahren, das Techniken aus der algorithmischen Geometrie verwendet, um die optimalen Intervallgrenzen zu finden. Der wesentliche Nachteil des vorgestellten Verfahrens liegt in der Beschränkung auf Regeln, welche nur aus 2 Items bestehen.

Die Autoren in [18] betrachten quantitativ-kategorische Assoziationsregeln. Das Interessantheitsmaß der Autoren basiert dabei auf dem Mittelwert von Transaktionen. Eine Regel wird dann als signifikant angesehen, wenn sich der Mittelwert über eine Teilmenge an Transaktionen in Relation zur Menge aller Transaktionen die nicht dieser Regel folgen, als signifikant herausstellt. Zur Berechnung des Signifikanzlevels ziehen sie dabei den Steiger-Z-Test heran. Als Null-Hypothese nehmen sie an, dass die Mittelwerte beider Teilmengen gleich sind. Wird diese Null-Hypothese abgelehnt (mit einer Konfidenz von 95%), so wird die ermittelte Regel als signifikant unterschiedlich von der Restmenge der Regeln angenommen.

In der Literatur finden sich auch Ansätze, welche versuchen, das Problem des quantitativen Assozoziation Rule Mining mit nicht-statistischen Verfahren zu lösen. So verwendeten die Autoren in [156, 182] zum Beispiel einen genetischen Algorithmus zur Problemlösung. Ein Individuum entspricht dabei einem k-Itemset, welches sich aus den Items und deren minimalen und maximalen Werten zusammensetzt. Mittels Crossover, Selektion und Mutation werden diese Items anschließend rekombiniert bzw. angepasst und deren Güte über eine Fitnessfunktion bestimmt.

5.4 Zusammenfassung

AnduIN bietet eine Vielzahl von Operatoren zur effizienten Verarbeitung und Auswertung von Daten sowohl aus herkömmlichen Datenströmen als auch aus Sensornetzwerken. Zusätzlich zu den grundlegenden Datenstrom-Operatoren bietet das System auch Unterstützung für die Verarbeitung von Anfragen mit räumlichem Bezug sowie Operatorimplementierungen komplexer Data-Mining-Verfahren. Im zurückliegenden Kapitel wurden exemplarisch die Integration verschiedener räumlicher Operatoren, ein adaptives Burst-Verfahren und eine Methode zum Finden häufiger Muster in Datenströmen über quantitativen Attributen vorgestellt. Zusätzlich zu der grundlegenden Arbeitsweise der Verfahren wurden deren Integration in *AnduIN* und notwendige Anpassungen für die (teilweise) In-Network-Verarbeitung präsentiert. Die korrekte Funktionsweise soll als Teil der Evaluation im nächsten Kapitel verifiziert werden.

Kapitel 6

Implementierung und Evaluation

Die in der Arbeit vorgestellten Techniken wurden als Teil von *AnduIN* umgesetzt [125]. Nachdem in Abschnitt 3 bereits die grundlegende Architektur des entwickelten Systems präsentiert wurde, soll im Abschnitt 6.1 auf Details der Implementierung eingegangen werden. Eine Evaluierung der in der Arbeit vorgestellten Konzepte und Techniken folgt anschließend in Abschnitt 6.2.

6.1 Implementierung

Die Implementierung von *AnduIN* basiert auf dem DSMS StreamDB [110,153], welches im Rahmen dieser Arbeit eine Vielzahl von Erweiterungen erfahren hat. Die Anfrageverarbeitung und die die Operatoren betreffenden Erweiterungen wurden in den letzten Kapiteln ausführlich beschrieben. Im Folgenden sollen zusätzliche Entwicklungen im Detail erläutert werden, welche die erweiterten Einsatzmöglichkeiten von *AnduIN* aufzeigen und für die anschließende Evaluierung von Interesse sind. Abschnitt 6.1 widmet sich hierbei der grundlegenden Client-Server-Architektur des Systems, Abschnitt 6.1.2 beschreibt die Möglichkeiten und die Umsetzung des grafischen Nutzerinterfaces, und Abschnitt 6.1.3 gibt einen detaillierten Überblick über die implementierte In-Network-Komponente des Systems und die verwendete Hardware-Plattform.

6.1.1 Client-Server-Architektur

Die Umsetzung von *AnduIN* als Client-Server-System ermöglicht eine plattformunabhängige Entwicklung von Clients und erlaubt somit eine einfache Integration der von *AnduIN* bereitgestellten Funktionen in bestehende Systeme. Neben dem durch den Server bereitgestellten konsolenbasierten Interface existieren gegenwärtig noch ein C++- und ein Java-Client. Die Clients selbst verbinden sich über TCP mit dem Server, so dass der Zugriff auch über das Internet möglich ist. Die Clients bilden im Wesentlichen eine einfache Nutzerschnittstelle zum System, d.h., sie bieten lediglich die Möglichkeit

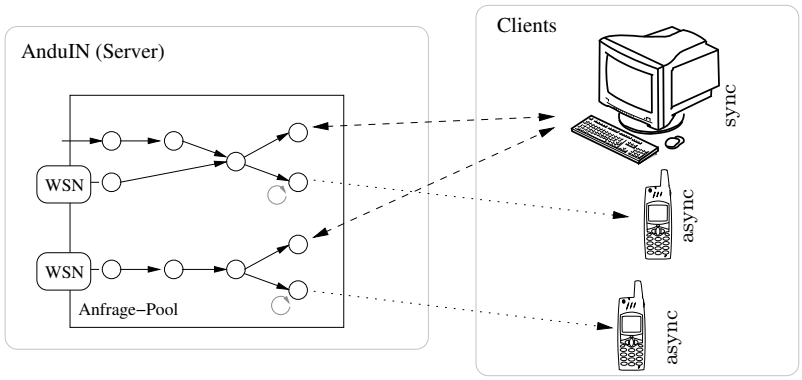


Abbildung 6.1: Client-Server-Ansatz von *AnduIN*

Anfragen zu definieren und Ergebnisse zu empfangen. Durch die Trennung von Client und Server ist es möglich, *AnduIN*-Clients auch auf leistungsschwacher Hardware wie zum Beispiel Mobiltelefonen einzusetzen (Abbildung 6.1).

Ein weiterer Vorteil des Client-Server-Ansatzes ist die Unterstützung eines Mehrbenutzerbetriebes. In *AnduIN* wird der Einfachheit halber jeder Client einem eigenständigen Systemnutzer zugeordnet. Jeder Client stellt also unabhängig von den anderen Clients Anfragen an das System. Ergebnisse werden ausschließlich an den Client weitergeleitet, welcher die Anfrage gestellt hat. Zusätzlich ist es aber auch möglich, dass sich Clients für Anfragen anderer Nutzer als Mithörende (*Listener*) registrieren. Sie erhalten dann auch für diese Anfragen entsprechende Ergebnisse [28].

Für die Registrierung der Clients beim Server wird das bereits existierende *publish/-subscribe*-Modell für Operatoren verwendet. Die Kommunikation zwischen Client und Server erfolgt dann über die Anfrage-Senke (wie in Abschnitt 4.1 beschrieben). Registriert sich ein Client für eine Anfrage, so wird dieser eine entsprechende Senke als Interface zum Client hinzugefügt.

Für die Verteilung der Ergebnisse an die Clients wurden zwei Modelle integriert:

- *Synchrone Kommunikation*: Der Server verteilt Ergebnisse direkt nach deren Fertigstellung an alle registrierten Clients. Hat ein Client zu diesem Zeitpunkt keine Verbindung zum Server aufgebaut, gehen diese Ergebnisse verloren. Die synchrone Kommunikation ist die Standardeinstellung für den Ergebnis-Nachrichtenversand zu Clients.

- *Asynchrone Kommunikation*: Ergebnisse werden vom Server in von den Clients definierten Ringpuffern zwischengespeichert. Der Client kann anschließend zu einem beliebigen Zeitpunkt die vollständige im Puffer enthaltene Ergebnismenge anfordern. Um einen Überlauf des Puffers zu verhindern, werden alte Ergebnisse kontinuierlich überschrieben. Da es sich bei dem Ringpuffer im Wesentlichen um eine Hauptspeicher-Datenbank handelt, kann dieser mittels entsprechender SQL-Kommandos vom Client abgefragt werden.

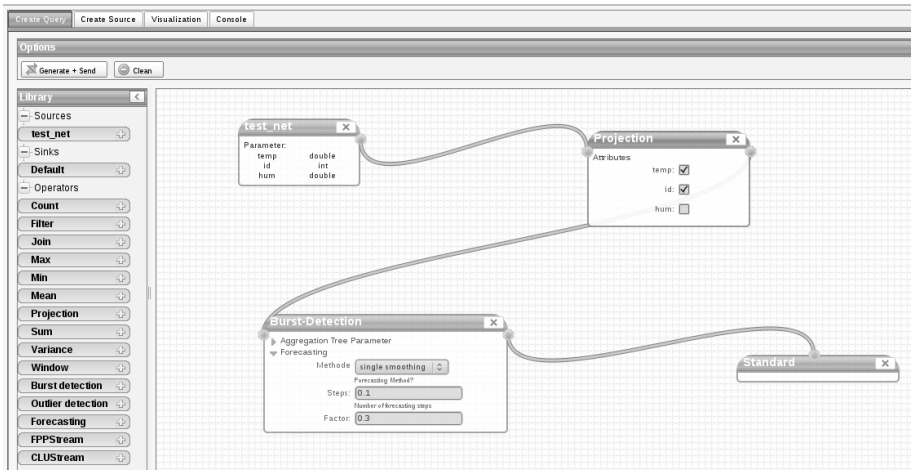


Abbildung 6.2: Visuelle Anfragebeschreibung

Im Anhang C werden die für die eben beschriebenen Aspekte notwendigen Erweiterungen der Anfragesprache im Detail erläutert.

6.1.2 Grafische Benutzeroberfläche

Neben der Möglichkeit, Anfragen über die Kommandozeile zu definieren, wurde zusätzlich ein GUI (*graphical user interface*) zur einfachen Anfragebeschreibung implementiert. Dieses wurde als Web-Applikation verwirklicht und in wesentlichen Teilen in JavaScript umgesetzt. Die Implementierung setzt dabei auf verschiedene externe Bibliotheken (WireIt[1], inputEX[2], YUI[3] u.a.) auf, welche die für die Darstellung benötigte Funktionalität bereitstellen. Neben der grafischen Anfragedefinition bietet die Oberfläche zusätzlich die Möglichkeit, Sensorknoten zu registrieren, Datenströme anzulegen (inklusive Netzwerkdatenströmen mit entsprechendem grafischen Anlegen des logischen Overlays) und Ergebnisströme zu visualisieren. Die verschiedenen Werkzeuge wurden in die Web-Oberfläche in Form von Reitern integriert, welche dem Anwender einen einfachen Zugriff auf die verschiedenen Funktionen ermöglichen.

Eine der wesentlichen Funktionen der Web-Oberfläche ist die unterstützte Anfragedefinition. Diese folgt dabei dem *Box-And-Arrow*-Prinzip, d.h., Operatoren werden als Boxen dargestellt, welche von den Daten des Stromes durchflossen werden (ähnlich dem Konzept der Anfragedefinition in Aurora [46]). Die Kommunikation zwischen den einzelnen Operatoren wird durch Verbindungslinien visualisiert. Der Anwender hat die Möglichkeit, aus einer Bibliothek von Operatoren zu wählen, welche die von AnduIN zur Verfügung gestellten Operatoren enthält. Diese können auf der Arbeitsfläche mit anderen Operatoren zu komplexen Anfragen kombiniert werden. Zusätzlich zu den Ope-

[1] http://javascript.neyric.com/wireit/
[2] http://neyric.github.com/inputex/
[3] http://developer.yahoo.com/yui/

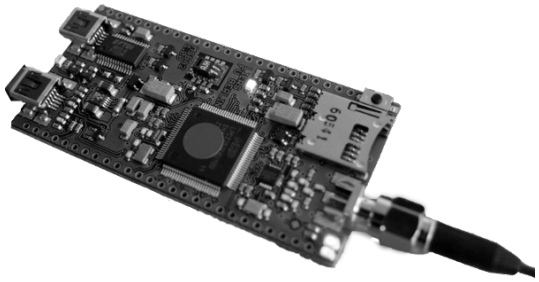


Abbildung 6.3: MSB-A2 Sensorknoten

ratoren stellt die Bibliothek eine Liste aller gegenwärtig verfügbaren Datenströme (inklusive Views) zur Verfügung. Das GUI folgt dabei in wesentlichen Teilen dem Konzept von Yahoo! Pipes[4], welches ähnliche Funktionen für die Komposition von Mashups bereitstellt. Abbildung 6.2 zeigt die GUI mit einer Beispielanfrage.

Die so vom Anwender erstellte Anfrage wird zum Zeitpunkt des Absendens von der GUI (auf Clientseite) analysiert und in eine Menge von CQL-Anfragen übersetzt, welche im Anschluss an den Server gesendet und von diesem ausgeführt werden. Die Anfrageanalyse auf der Clientseite beschränkt sich dabei im Wesentlichen auf eine Korrektheitsprüfung der einzelnen Boxen und Pfeile und das Übersetzen in CQL-Anfragen. Hierbei werden die einzelnen Knoten des Graphen in Post-Order-Reihenfolge durchlaufen und jede Box in eine einzelne Sicht übersetzt. Die dabei entstehenden Sichten sind jeweils eindeutig (über einen zusätzlichen Zeitstempel) und werden dem Anwender nicht präsentiert. Dieser sieht nach dem Anlegen lediglich seine definierte Anfrage.

Die Visualisierungskomponente wurde in Form von Java-Applets umgesetzt, welche den vom Server zur Verfügung gestellten Datenstrom analysieren und in Abhängigkeit der Nutzerdefinition visualisieren. In Anhang B finden sich Abbildungen zur entwickelten Visualisierungskomponente.

6.1.3 In-Network Query-Prozessor

Die Entwicklung der In-Network-Komponente von *AnduIN* erfolgte auf Basis einer verhältnismäßig leistungsstarken ARM-Plattform, welche im folgenden Abschnitt beschrieben werden soll. Da für eine effiziente Entwicklung zum Zeitpunkt der Arbeit leider kein Netzwerksimulator zur Verfügung stand, musste eine entsprechende Simulationssumgebung entwickelt werden, welche anschließend kurz beschrieben wird.

6.1.3.1 Sensorknoten – Plattform

Die Implementierung der INQP-Komponente von *AnduIN* basiert auf Sensorknoten der MSB-A2-Plattform der FU Berlin. Für die Entwicklung standen acht Sensorknoten

[4]http://pipes.yahoo.com

(siehe Abbildung 6.3) mit folgender (identischer) Hardware-Ausstattung zur Verfügung:

- MSBA2-Boards mit ARM LPC2387 CPU
- CC1100 Funk-Modul
- 512KB Flashspeicher und 98 KB RAM
- SHT11-Sensoren (Luftfeuchtigkeit und Temperatur)
- 2 USB-Ports (1x serielle Programmierung und 1x PC-Verbindung)

Zusätzlich zu den On-Board-Sensoren wurde pro Sensorknoten ein zusätzliches Kameramodul installiert, welches die Aufnahme von Videos und Bildern sowie eine einfache Bewegungsdetektion erlaubt. Jedes der angeschlossenen Geräte (inklusive der auf den Boards installierten LED's) trägt, insofern die entsprechende Hardware nicht explizit deaktiviert wurde, zum Stromverbrauch der Sensorknoten bei.

Als Betriebssystem diente das von der FU Berlin entwickelte *Feuerware*[5] in Version 0.4, welches ursprünglich ein Contiki-Derivat war. Das Betriebssystem stellt dabei die wesentlichen Treiber für die Hardwarekomponenten sowie zwei Timer zur Verfügung. Des Weiteren wurde mit dem Micro-Mesh-Routing-Protokoll [107] ein von der FU Berlin entwickeltes eigenes Routing-Protokoll in das Betriebssystem integriert.

6.1.3.2 Simulationsumgebung

Zum Zweck der Entwicklung und Evaluierung von verteilten Algorithmen für WSN werden üblicherweise Netzwerksimulatoren wie TOSSIM [141] oder OMNeT [67] verwendet. Aufgrund der großen Abweichungen zwischen Contiki und der eingesetzten Feuerware-Version war die Verwendung des ursprünglich mit Contiki ausgelieferten Netzwerksimulators *netsim* leider nicht möglich. Aus diesem Grund wurde im Rahmen der Entwicklung eine sehr einfache eigene Simulationsumgebung geschaffen.

Zu diesem Zweck wurden alle für den INQP benötigten und vom Betriebssystem zur Verfügung gestellten Funktionen soweit gekapselt, dass sie durch eine entsprechende Implementierung der Simulationsumgebung ausgetauscht werden konnten. Hierzu wurden im Wesentlichen die physischen Sensoren, die Timer, die Interprozesskommunikation, die Netzwerkkommunikation und die Speicherverwaltung durch entsprechende Simulatorimplementierungen ersetzt. Die Laufzeitumgebungen der einzelnen Knoten entsprechen in der Simulationsumgebung anschließend je einer eigenen Programminstanz. Die so simulierten Knoten tauschen über UDP Nachrichten untereinander aus. Auf die Nachbildung des Routingprotokolls in der Simulationsumgebung wurde bei der Implementierung komplett verzichtet. Wurde den simulierten Sensorknoten manuell keine Netzwerk-Topologie aufgeprägt, so kommunizierten diese immer direkt mit dem Knoten der Basisstation bzw. direkt untereinander. Der Verzicht auf die Routingschicht

[5] http://cst.mi.fu-berlin.de/projects/ScatterWeb/documentation/FeuerWare/trunk/manual/

stellt im Rahmen dieser Arbeit keine wesentliche Einschränkung dar, da sich sämtliche hier entwickelten Verfahren oberhalb der Routingebene bewegen. Zudem kann bei der Simulation von Topologien auf deren manuelle Definition zurückgegriffen werden.

Trotz der Einfachheit der Simulationsumgebung bietet diese grundlegende Funktionen zur Evaluierung von Algorithmen. So lassen sich neben der korrekten Funktionsweise von Operatoren auch andere Informationen, wie zum Beispiel der Speicherverbrauch der Anwendung und die für die Kommunikation auf Basis des definierten Overlays notwendige Nachrichtenanzahl, exakt ermitteln. Ein weiterer Vorteil der Simulationsumgebung ist, dass Messwerte exakt vorgegeben werden können. So lässt sich der Input für die einzelnen Operatoren genau bestimmen. Dies ist insbesondere für die Evaluierung der Operatoren sehr hilfreich. Darüber hinaus lässt sich mit Hilfe der Simulationsumgebung die für die Anfrageausführung benötigte Zeit aufgrund des simulierten Timers deutlich verkürzen.

6.2 Evaluierung

Im folgenden Kapitel sollen die in der Arbeit präsentierten Verfahren und Lösungsansätze evaluiert werden. Die Evaluierung gliedert sich dabei in zwei Abschnitte. Im ersten Teil sollen verschiedene Aspekte der Anfrageoptimierung, insbesondere das vorgeschlagene Kostenmodell betrachtet werden. Im zweiten Teil werden die in Verbindung mit *AnduIN* eingeführten Operatoren hinsichtlich ihrer korrekten Funktionsweise bzw. ihrer Performance analysiert.

6.2.1 Anfrageoptimierung

Im folgenden Abschnitt werden verschiedene Aspekte der Anfrageoptimierung evaluiert. Es soll zunächst gezeigt werden, dass die Wahl der Operatorparameter und verschiedene Datencharakteristika Einfluss auf die Energiebilanz eines Anfrageplanes haben. Anschließend wird dargestellt, wie die für das in Abschnitt 4.2 beschriebene Kostenmodell benötigten Operatorkosten ermittelt werden können. Auf Grundlage der so ermittelten Kosten wird anschließend exemplarisch die Gültigkeit des Kostenmodells gezeigt.

6.2.1.1 Einsatz alternativer Operatorimplementierungen

Bereits aus dem Bereich der DBMS ist bekannt, dass die Kosten für die Ausführung eines verwendeten physischen Operators sowohl von den Charakteristika der verarbeiteten Daten als auch von der Wahl der Operatorparameter (zum Beispiel Filterprädikaten) abhängen. Die Berücksichtigung dieser Tatsache ist insbesondere im Kontext von drahtlosen Sensornetzwerken wichtig, da die richtige Auswahl einer speziellen Operatorimplementierung signifikanten Einfluss auf die Lebenszeit des Netzwerkes haben kann. Die

Gültigkeit obiger Aussage soll im Weiteren gezeigt werden.

These 1 *Die Wahl der physischen Implementierung eines Operators und die Datencharakteristika haben einen signifikanten Einfluss auf den Energieverbrauch eines WSN und damit auf dessen Lebensdauer.*

In [73, 74] wurde ein Ansatz zur Erkennung von Regionen, in denen Sensorknoten anormale Messwerte erfassen, präsentiert. Ein typisches Einsatzszenario für ein solches Verfahren ist die Erfassung und Meldung von Waldbränden und deren räumliche Ausdehnung. In den genannten Arbeiten wurden verschiedene verteilte Varianten des Algorithmus für die Ermittlung der Regionen vorgestellt. Darüber hinaus wurden Kostenabschätzungen für die einzelnen Varianten des Verfahrens gegeben. Im Folgenden soll anhand dieses Verfahrens gezeigt werden, welchen Einfluss verschiedene Datencharakteristika und Operatorparameter auf die Energiebilanz der möglichen physischen Implementierungen haben. Gegeben seien die folgende Sensornetzwerkquelle

```
CREATE STREAM mystream
  (time int, id int, temp double, hum double)
  NETWORK
  SAMPLE 1 SECONDS
```

und die Anfrage:

```
SELECT
   time, region
FROM
  (SELECT
    id, time, temp
   FROM mystream [outlier(exclude => id, exclude => time)]
  )[regiondet()]
```

Die Sensornetzwerkquelle *mystream* definiert einen Datenstrom, welcher die Attribute Zeit (*time*), ID (*id*), Temperatur (*temp*) und Luftfeuchtigkeit (*hum*) bereitstellt. Die Datenerfassung auf den Sensorknoten soll dabei im Abstand von jeweils 1 Sekunde erfolgen.

Basierend auf dieser Datenstromquelle wird die Anfrage für die Ausreißerdetektion definiert, wobei die Attribute *id* und *time* von der Ausreißerdetektion ausgenommen wurden. Ergebnis der inneren Anfrage ist ein Datenstrom, welcher lediglich Ausreißer enthält. Auf Grundlage dieses Datenstromes erfolgt im Weiteren die Bestimmung der Regionen (*regiondet*) mit anormalen Messwerten.

Auf die Angabe von Parametern für die einzelnen Synopsen wurde der Übersichtlichkeit halber weitestgehend verzichtet. *AnduIN* erstellt auf Grundlage dieser Anfrage drei alternative Ausführungspläne $\bar{q}_{central}, \bar{q}_{outlier}$ und $\bar{q}_{lead}$ (siehe Abbildung 6.4). Im Fall von $\bar{q}_{central}$ finden sämtliche Berechnungen auf der zentralen Instanz statt. Bei $\bar{q}_{outlier}$ erfolgt die Ausreißererkennung im WSN, so dass hier in Abhängigkeit von der Ausreißerhäufigkeit eine Nachrichtenreduktion erzielt wird. Die Regionenbestimmung wird

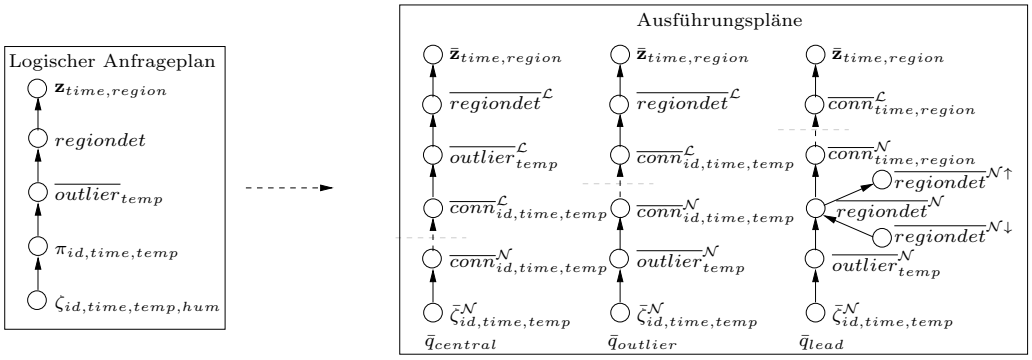
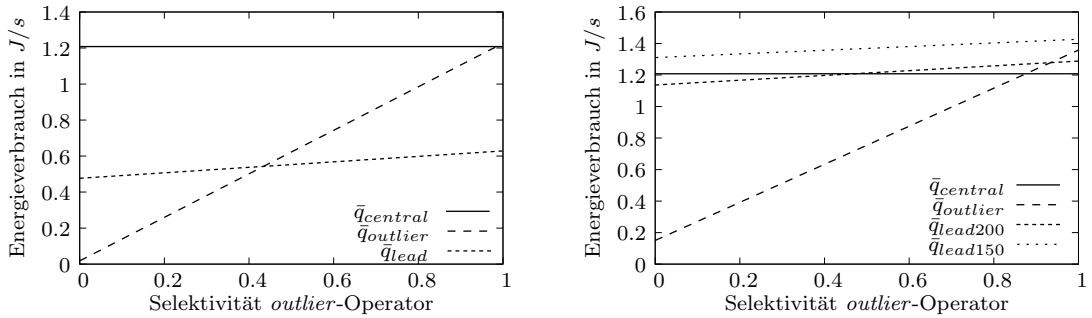


Abbildung 6.4: Logischer Anfrageplan und mögliche physische Ausführungspläne für die Beispielanfrage



(a) $w = 600$ (b) $w = 5000$

Abbildung 6.5: geschätzte Ausführungskosten für unterschiedliche Selektivitäten des *outlier*-Operators ($m = 1000, m_l = 200(150), h = 3, r_m = 1$)

auf der zentralen Instanz durchgeführt. Der Anfrageplan $\bar{q}_{lead}$ wird komplett im WSN verarbeitet, d.h., sowohl die Ausreißererkennung als auch die Regionenbestimmung werden innerhalb des WSN durchgeführt. Lediglich die detektierten Regionen werden als Ergebnis der In-Network-Anfrage an die zentrale Instanz übermittelt. Die Regionenbestimmung basiert auf einem Leader-Ansatz. Ein Leader ermittelt hierbei die Regionen für die Sensoren in seiner Nachbarschaft und tauscht diese Informationen mit anderen Leaderknoten aus, um die Ergebnisregionen zu erzeugen. Der *outlier*-Operator wird vollständig auf den Sensorknoten ausgeführt.

Die Abbildungen 6.5(a) und 6.5(b) zeigen den nach [74] geschätzten Energieverbrauch der drei möglichen Ausführungspläne für verschiedene Selektivitäten des Ausreißeroperators und unterschiedliche Fenstergrößen w. Unterschiedliche Selektivitäten repräsentieren dabei unterschiedliche Datencharakteristika. Für die Abschätzung wurde ein Sensornetzwerk mit 10a00 Knoten und einer mittleren Höhe von $h = 3$ angenommen. Für die Anfrage $\bar{q}_{lead150}$ wurde ein Sensornetz mit 150 Leaderknoten angenommen, für $\bar{q}_{lead200}$ ein WSN mit 200 Leaderknoten.

Die Abbildungen zeigen, dass eine generelle In-Network-Verarbeitung nicht in jedem

Operator	Zeit in ms	elektrische Stromstärke in mA	konsumierte Energie in μJ
Sampling (Temperatur)	264.5	4.3	3753.4
Sampling (Luftfeuchtigkeit)	114.0	4.4	1655.3
Sampling (Temperatur u. Luftfeuchtigkeit)	359.0	4.0	4738.8
Daten Senden (Initialisierung)	2.3	7.9	61.7
Daten Senden (62Byte - 1 Push)	3.6	8.4	99.7
Daten Senden (Initialisierung + 73xPush)	271.0	-	7344.8
Ausreißererkennung	6.1	5.5	110.7

Tabelle 6.1: Gemessene Aktivierungsenergien für ausgewählte In-Network-Operatoren

Fall die energetisch beste Lösung darstellt. Im Fall der Ausreißerdetektion über einem Fenster der Größe $w = 600$ ist ab einer Selektivität von ca. 0.4 eine zentrale Berechnung der Regionen effizienter als die In-Network-Berechnung (Abbildung 6.5(a)). Aufgrund der hohen Anzahl an detektierten Ausreißern ist eine häufige Aktualisierung der Regionen notwendig. Dies resultiert wiederum in einer entsprechend hohen Anzahl von Nachrichten für das Propagieren der Regionen zwischen den Leaderknoten. Die In-Network-Verarbeitung wird also mit steigender Ausreißerrate zunehmend ineffizient. Abbildung 6.5(b) zeigt den Energieverbrauch der verschiedenen Implementierungen für eine Fenstergröße von $w = 5000$. Ab einer Selektivität von ca. 0.85 ist es in diesem Fall sogar effizienter, vollständig auf eine In-Network-Verarbeitung zu verzichten, da im Fall einer solch hohen Ausreißerrate nahezu alle Datensätze von den Sensorknoten zur zentralen Instanz gesendet werden müssen. Zusätzliche Berechnungen auf den Knoten führen hierbei nur zu einer erhöhten Last, welche nicht durch eine Reduktion der benötigten Nachrichten ausgeglichen wird.

Diese simulierten Ergebnisse zeigen bereits, dass der Energieverbrauch des WSN durch die geeignete Wahl alternativer verteilter Operatorimplementierungen erheblich reduziert werden kann. Diese Feststellung ist eines der wesentlichen Motive für den in der Arbeit vorgeschlagenen Aufbau der Laufzeitumgebung, welcher die Integration einer Vielzahl an alternativen Operatoren erlaubt. Des Weiteren zeigen die Ergebnisse, dass insbesondere im Kontext einer kontinuierlichen Anfrageverarbeitung eine regelmäßige Überprüfung der datenabhängigen Operatorcharakeristika wie zum Beispiel der Operatorselektivität empfehlenswert ist. Sofern nötig, können laufende Ausführungspläne durch alternative Pläne ersetzt werden.

6.2.1.2 Operatorkosten

In Abschnitt 4.2 wurden die beiden Größen Energiekosten im WSN und Datendurchsatz auf der zentralen Komponente für die Bewertung der physischen Ausführungspläne herangezogen. Für die Berechnung der Kosten eines Ausführungsplanes werden dabei die Kosten für die Aktivierung eines Operators und dessen Selektivität benötigt. Es

wurde argumentiert, dass sich erstere entweder durch Referenzmessungen (bei Operatoren mit konstantem Berechnungsaufwand) oder durch eine auf einer Referenzmessung basierende Approximation bestimmen lassen. Im Weiteren soll diese These verifiziert werden:

These 2 *Es ist möglich, die Energiekosten für die einmalige Aktivierung eines gegebenen Operators mit konstantem Berechnungsaufwand im WSN durch entsprechende Referenzmessungen zu bestimmen. Weiterhin ist es möglich, die Aktivierungsdauer für einen gegebenen Operator mit konstantem Berechnungsaufwand auf der zentralen Instanz durch Referenzmessungen zu bestimmen.*

Im Folgenden sollen die hierfür notwendigen Untersuchungen beschrieben werden.

Energiekosten pro Operator Die Vermessung der Operatoren erfolgte auf Grundlage des in Abschnitt 6.1.3 beschriebenen batteriebetriebenen Sensorknotens, welcher von einem Oszilloskop überwacht wurde. Prinzipiell bietet der verwendete Sensorknoten auch die Möglichkeit der Energieüberwachung über einen internen Coulomb-Zähler (LTWQ e3). Allerdings haben Testmessungen gezeigt, dass der Coulomb-Zähler große Ungenauigkeiten aufweist. Im Fall von Operatoren, welche häufig nur wenige Microsekunden aktiv sind, würden dadurch die Ergebnisse stark verfälscht werden. Zudem ist das Auslesen und Weiterleiten der gemessenen Werte mit einem zusätzlichen Energieverbrauch verbunden, welche bei On-Board-Messungen nicht auf einfache Weise von den eigentlichen Aktivierungskosten des Operators separiert werden könnten.

Um möglichst genaue Messungen durchführen zu können, wurde daher ein externes Oszilloskop für die Messungen eingesetzt. Das Oszilloskop erfasste dabei den Spannungsabfall über einen für die Messungen zwischen der Batterie und dem Board des Sensorknotens eingebrachten Widerstand mit 1Ω (der Widerstand wies eine Toleranz von ca. 3% auf). Auf diese Weise konnte der auf die Ausführung der Operatoren zurückzuführende Stromverbrauch und damit die für die Ausführung benötigte Energie berechnet werden. Das Board selbst benötigt eine Nominalspannung von $3.3V$. Da die Batterie eine Spannung von ca. $4.3V$ zur Verfügung stellte, musste durch das Einfügen des Widerstands keine relevante Beeinträchtigung der Messergebnisse befürchtet werden.

Das Oszilloskop erfasste die Spannung mit einer Frequenz von $20kHz$. Da die gemessenen Stromverbräuche sowohl von der möglichst exakten Bestimmung des Start- und Endzeitpunktes für die Operatorausführung als auch von der Samplingrate des Oszilloskops abhängt, wurden die Messungen wiederholt durchgeführt und anschließend gemittelt. Tabelle 6.1 stellt einen Auszug der so für die einzelnen Operatoren ermittelten Energieverbräuche dar. Die meisten Messungen konnten auf Basis eines einzelnen Sensorknotens durchgeführt werden. Für das Vermessen der für das Senden und Empfangen benötigten Energie wurde ein zweiter Knoten, welcher für den Kommunikationsaufbau mit dem überwachten Knoten verantwortlich war, hinzugefügt.

Prinzipiell ist der Energieverbrauch der Sensorknoten im Sleep-Mode für die Kostenermittlung nach dem beschriebenen Kostenmodell irrelevant; zu Vergleichszwecken wurde dieser dennoch bestimmt. In der Ruhephase floss im Mittel ein Strom von $3.73mA$, was

	Fenstergröße		
Operator	1	10	100
Projektion	126	-	-
Filter	136	-	-
Aggregation (Min/Max)	-	170 + 332	166 + 367
Grouping	-	139 + 909	141 + 956
Nested Loop Join	-	734 + 1211 + 1220	6035 + 9031 + 8657
Hash Join	-	342+875+817	1225 + 3654 + 3600
Ausreißer Detektion	-	178	393

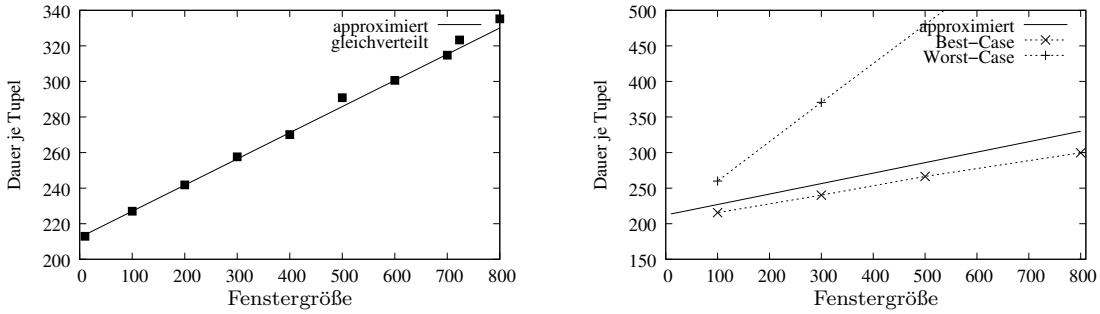
Tabelle 6.2: Berechnungszeit pro Tupel (Zeit pro Tupel in μs) für ausgewählte DSMS-Operatoren

im Vergleich zu anderen Sensorknoten verhältnismäßig hoch ist. Mögliche Ursachen hierfür sind in der vergleichsweise leistungsstarken Hardware der Sensorknoten, den zusätzlich angeschlossenen externen Komponenten wie zum Beispiel der Kamera und dem aktivierten SD-Leser, welcher für die Konfiguration der *AnduIN*-Laufzeitumgebung notwendig ist, zu suchen.

Durchsatz der Operatoren auf der zentralen Instanz Um die Menge an Tupeln, die ein Operator pro Zeiteinheit auf der zentralen Instanz verarbeiten kann, zu bestimmen, wurde eine Statistik-Komponente in die DSMS-Komponente von *AnduIN* integriert. Der Statistik-Monitor erfasst dabei für jeden Operator die Menge der eingehenden und ausgehenden Tupel sowie die Zeit die Aktivierungdauer des Operators. Auf Grundlage der erfassten Gesamtaktivierungszeit eines Operators und der Anzahl der von ihm verarbeiteten Datensätze in diesem Zeitraum lässt sich die Bearbeitungsdauer für ein einzelnes Tupel ermitteln.

Wie bereits erwähnt, ist die Menge an verarbeiteten Tupeln pro Zeiteinheit bzw. die Verweildauer eines einzelnen Datensatzes in einem Operator direkt von dem für die Messwertermittlung verwendeten System abhängig. Daher wurden alle durchgeführten Messungen auf ein und demselben System durchgeführt (AMD 2GHz, 2GB RAM, Debian GNU/Linux 5.0 mit gcc-4.4). Die so ermittelten relativen Verarbeitungszeiten können dann auf verschiedenen Systemen zum Einsatz kommen.

Tabelle 6.2 zeigt die Messergebnisse für ausgewählte Operatoren. Die Tabelle zeigt, dass Projektion und Filter wie erwartet die schnellsten Operatoren sind. Im Fall fensterbasierter Operatoren setzt sich die Verarbeitungszeit aus der Zeit für die Analyse der Synopse und der für die eigentliche Berechnung benötigten Zeit zusammen. Die einzelnen Kosten werden in Tabelle 6.2 durch die einzelnen Werte in einer Zelle repräsentiert. Der erste Wert entspricht der für die Berechnungen benötigten Zeit, die weiteren Kosten sind mit der Verarbeitung der Fenster verbunden. Die Fensterfunktionalität wurde in *AnduIN* gekapselt, so dass getrennte Messungen ohne weiteres möglich waren. Im Fall des Verbundes handelt es sich um zwei Fenster.



(a) (b)

Abbildung 6.6: Approximierte und reale Ausführungskosten für verschiedene Fenstergrößen am Beispiel des Minimum-Operators

Approximation der Operatorkosten Betrachtet man Operatoren mit nicht-konstantem Berechnungsaufwand, so müssen durch diese verursachte Kosten in Abhängigkeit eines oder mehrerer Parameter approximiert werden. Im Folgenden soll daher nachstehende These verifiziert werden:

These 3 *Die Kosten eines Operators mit nicht-konstantem Berechnungsaufwand können hinreichend genau approximiert werden, sofern die Datencharakteristika der zu verarbeitenden Daten bekannt sind.*

In Abschnitt 4.2.1.2 wurde exemplarisch eine Approximation für die durch den Minimum-Operator verursachten Energiekosten betrachtet. Im Weiteren soll dieses Beispiel mit entsprechenden Messungen unterlegt werden.

Da sich die Verarbeitungskosten auf der zentralen Komponente sowohl einfacher als auch genauer bestimmen lassen, wurde das Experiment auf Basis des Datendurchsatzes durchgeführt. Die Kostenapproximation für einen Minimum-Operator erfolgte in Abhängigkeit von der Fenstergröße, wobei diese als die Anzahl von Elementen pro Fenster definiert wurde. Für die Approximation der Kosten wurden Messungen mit Fenstern der Größe 10 und 100 durchgeführt. Die für die Messung erzeugten Daten waren dabei gleich verteilt über dem Intervall $[0, 100]$. Auf Basis der so ermittelten Werte wurde anschließend eine lineare Abschätzung des Operatordurchsatzes in Abhängigkeit von der Fenstergröße vorgenommen (Abbildung 6.6, durchgehende Linie). Zusätzlich wurden Messungen für verschiedene Fenstergrößen vorgenommen, um die Güte der Approximation zu überprüfen. Abbildung 6.6(a) zeigt, dass im Fall gleich verteilter Daten die lineare Approximation sehr gute Ergebnisse bezüglich der Genauigkeit erzielt.

Um Abweichung bei nicht gleich verteilten Daten zu bestimmen, werden im Folgenden die beiden Extremfälle betrachtet, welche beim Minimum-Operator auftreten können: (i) bei der Durchsuchung des Fensters wird das gewünschte Objekt immer sofort gefunden (Best-Case) bzw. (ii) das verwendete Fenster muss in jedem Fall vollständig untersucht werden (Worst-Case). Um beide Fälle zu simulieren, wurden Daten mit stetig steigenden bzw. fallenden Werten erzeugt. Die Ergebnisse zeigen, dass es prin-

zipiell zu starken Abweichungen gegenüber den approximierten Werten kommen kann. Insbesondere im Worst-Case-Szenario ist die wiederkehrende vollständige Prüfung des Fenster mit deutlich höheren als den geschätzten Kosten verbunden. Die Abweichung steigt dabei in Abhängigkeit der Fenstergröße.

Das Best-Case-Szenario zeigt Überraschendes (siehe Abbildung 6.6(b)): Hier hätte man erwartet, dass die Kosten auf einem konstanten Niveau bleiben, da der gesuchte Wert bereits beim Test des ersten Elements im Fenster gefunden wird. Es zeigt sich hier jedoch ein linearer Trend.

Bereits dieses einfache Beispiel zeigt zwei wesentliche Punkte: (i) Eine gute Approximation der Kosten für die Ausführung der Operatoren ist möglich und (ii) die Datenverteilung hat auch hier einen nicht zu vernachlässigenden Einfluss auf die Kosten. Sofern es nicht kontinuierlich zu den Extremfällen kommt, kann die Approximation hinreichend gute Ergebnisse liefern.

6.2.1.3 Kostenmodell

Einer der Schwerpunkte dieser Arbeit war die Entwicklung des Kostenmodells. Dass die Verwendung von Operatorselektivitäten für die Kostenabschätzung in DSMS geeignet ist, wurde bereits hinreichend untersucht [45, 207, 215]. Das vorgestellte Kostenmodell dehnt den bekannten Ansatz auf drahtlose Sensornetzwerke aus, so dass eine plattformübergreifende Anfrageoptimierung möglich wird. In Abschnitt 4.2.1.4 wurde gezeigt, dass der vorgestellte selektivitätsbasierte Ansatz geeignet ist, um bisherige In-Network-Kostenmodelle abzubilden. Im Weiteren soll daher lediglich gezeigt werden, dass mit Hilfe der oben bestimmten Aktivierungskosten für die einzelnen Operatoren und der bekannten Selektivitäten eine korrekte Bewertung von Ausführungsplänen im WSN möglich ist. Konkret soll nachstehende These betrachtet werden:

These 4 *Das in Abschnitt 4.2 präsentierte Kostenmodell liefert hinreichend genaue Ergebnisse, um den kostengünstigsten Ausführungsplan zu ermitteln.*

Hierzu sollen für einen Anfrageplan approximierte Kosten mit den auf einem Sensorknoten tatsächlich gemessenen Anfragekosten verglichen werden.

Wie in Abbildung 4.4 auf Seite 55 zu sehen ist, ergeben sich die Gesamtkosten für die Ausführung des In-Network-Teils einer Anfrage durch die lineare Kombination der Kosten der einzelnen Verarbeitungsphasen (Operatoren). Der Vergleich zwischen approximierten Kosten und tatsächlichen Kosten soll daher im Folgenden anhand einer repräsentativen Anfrage gezeigt werden. Diese umfasst alle wesentlichen Aspekte einer typischen In-Network-Anfrage: Die Anfrage sampelt eine Menge von Werten, verarbeitet diese und sendet Ergebnisse an die Basisstation. Zudem sollen zwei innere Operatoren mit einem vollkommen unterschiedlichen Energieverbrauch eingesetzt werden.

Es wurde folgende CQL-Anfrage verwendet:

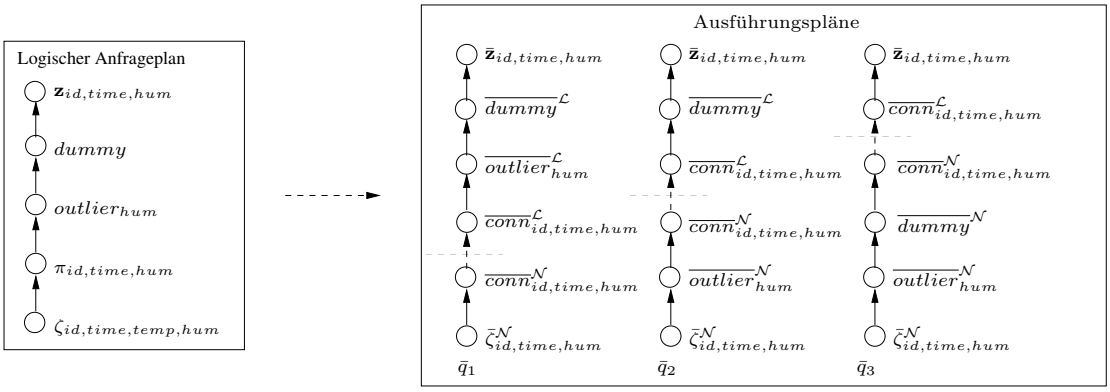


Abbildung 6.7: Logischer Anfrageplan und mögliche physische Ausführungspläne für die Beispielanfrage

```
SELECT id, time, hum
FROM
  (SELECT
    id, time, hum
    FROM mystream [outlier (win=>10)]
  ) [ dummy() ]
```

mystream bezieht sich auf die Datenstrom-Definition aus Abschnitt 6.2.1.1, wobei ein Sampling-Intervall von zwei Sekunden verwendet wurde. Der Grund hierfür liegt in dem von den Sensorknoten verwendeten Routing-Protokoll. Zwar werden im Mittel für das Versenden eines einzelnen Datenpaketes lediglich $267ms$ benötigt, im äußersten Fall kann das Senden aber auch bis zu $532ms$ in Anspruch nehmen. Zusammen mit der für die Operatoreverarbeitung benötigten Ausführungszeit kann es somit unter Umständen zur Überlagerung von aktiven Anfrageteilen kommen. Der Sensorknoten würde eine aktive Anfrage zu Ende bearbeiten und eine eventuell überlagernde neue Aktivierung auslassen. Um dies zu verhindern, wurde das Sampling-Intervall entsprechend angepasst.

In der inneren Anfrage wird eine Ausreißerdetektion über dem Attribut *hum* (Luftfeuchtigkeit) durchgeführt (Parameter für die komplexen Operatoren wurden der Übersichtlichkeit halber weitestgehend weggelassen). Wird ein Ausreißer gefunden, so wird dieser von einem $\overline{dummy}$-Operator verarbeitet. Das Ergebnis wird anschließend an die Basis-Station gesendet. Der $\overline{dummy}$-Operator ist ein Blackbox-Operator, dessen Eigenschaften (Kosten und Selektivität) für die Experimente gezielt definiert werden konnten. Für das Experiment wurde mit Energiekosten von $3971.9\mu J$ pro Aktivierung (Aktivierungszeit von $118ms$ bei durchschnittlich $10.2mA$) gearbeitet. Der für die Beispielanfrage von *AnduIN* erzeugte logische Anfrageplan und dessen mögliche physische Ausführungspläne sind in Abbildung 6.7 zu sehen.

Da gemäß des in der Arbeit vorgestellten Ansatzes auf allen Knoten die gleiche Laufzeitumgebung zum Einsatz kam, war das Vermessen eines einzelnen Sensorknotens ausreichend. Lediglich eventuell notwendige Kosten für den Datentransfer über mehrere

Ausführungs-	geschätzte Kosten			gemessene Kosten
plan	Verarbeitung	Sleep	Summe	
$\bar{q}_1$	0.27000	0.59686	0.86686	0.92528
$\bar{q}_2$	0.16315	0.64382	0.80697	0.86816
$\bar{q}_3$	0.14892	0.65589	0.80481	0.85265

Tabelle 6.3: geschätzter vs. gemessener Energieverbrauch pro Minute (in J).

Hops konnten so nicht bestimmt werden. Die Messungen wurden wiederum mittels eines Oszilloskopes durchgeführt. Der vermessene Sensorknoten war batteriebetrieben und sendete seine Ergebnisse direkt an den Gate-Knoten, welcher diese an das DSMS weiterleitete.

Die Messungen mittels Oszilloskop wurden bei $10kHz$ vorgenommen. Jede Anfrage wurde mindestens eine Minute ausgeführt. Anschließend wurde der mittlere Stromverbrauch ermittelt, welcher für die Berechnung des Stromverbrauchs pro Minute diente.

Für die Abschätzung der Energiekosten mittels des Kostenmodells wurden die folgenden Selektivitäten angenommen: $\sigma(\overline{outlier}_{hum}) = 0.5$ und $\sigma(\overline{dummy}) = 0.33$. Die Selektivität für den $\overline{dummy}$-Operator konnte für die Anfrageausführung manuell definiert werden. Die Selektivität für $\overline{outlier}_{hum}$ wurde nachträglich aus dem Messergebnis ermittelt.

Tabelle 6.3 zeigt die geschätzten und die tatsächlich gemessenen Ausführungskosten im Vergleich. Es ist zu sehen, dass sich die gemessenen und approximierten Kosten für die Verarbeitung deutlich voneinander unterscheiden. Der Grund hierfür liegt auf der Hand: Im Kostenmodell wurden keinerlei Kosten für den Sleep-Mode berücksichtigt. Im Weiteren wird gezeigt, wie die Sleep-Kosten aus dem beschriebenen Kostenmodell abgeleitet werden können. Die so ermittelten Werte finden sich in der dritten Spalte von Tabelle 6.3.

Für jeden Operator sind sowohl dessen Aktivierungshäufigkeiten bekannt (Formel 4.1) als auch die für die einmalige Ausführung benötigte Zeit. Basierend auf diesen Werten kann ermittelt werden, über welchen Zeitraum ein Sensorknoten aktiv bzw. inaktiv ist. Weiterhin wird für die Berechnung angenommen, dass sich ein Sensorknoten immer genau dann im Sleep-Mode befindet, wenn er nicht aktiv ist. Somit kann auf Grundlage der aktiven Zeit eines Knotens die Zeit ermittelt werden, die ein Knoten im Sleep-Mode verbringt. Ist diese bekannt, dann lässt sich der Energieverbrauch entsprechend ermitteln:

$$\mathsf{E}_{sleep}(\bar{q}, \Delta t) = W_{sleep} \cdot \left(\Delta t \cdot N - \sum_{\bar{op} \in \bar{q}} \mathsf{T}(\bar{op}) \cdot \phi(\bar{q}, \bar{op})_{\Delta t} \right)$$

N entspricht der Anzahl an Knoten im WSN. $\mathsf{T}(\bar{op})$ bezeichnet die für die Abarbeitung eines Operators $\bar{op}$ benötigte Zeit. W_{sleep} entspricht der elektrischen Leistung im Sleep-Mode. Im Beispiel ist $W_{sleep} = 3.73mA \cdot 3.3V$.

Die Ergebnisse in Tabelle 6.3 zeigen, dass durch das in der Arbeit vorgeschlagene Kostenmodell die tatsächlichen Energiekosten unter Einbeziehung der Kosten für den Sleep-Modus verhältnismäßig gut approximiert werden können (sofern alle notwendigen Parameter bekannt sind). Die Abweichungen zwischen den tatsächlichen Kosten und den geschätzten Kosten (inklusive der Sleep-Kosten) von ca. 6-7% resultieren vermutlich aus Messungenauigkeiten. Kleinere Messfehler summieren sich entsprechend auf. Insbesondere Messfehler bei den Sleep-Kosten können die Approximation stark beeinflussen.

Wie bereits bei der Beschreibung des Kostenmodells argumentiert wurde, ist das Ziel der Kostenberechnung nicht eine möglichst genaue Approximation des Energiekonsums der einzelnen Anfragepläne, sondern die Feststellung der relativen Ordnung dieser bezüglich ihrer tatsächlichen Ausführungskosten. Obiges Beispiel zeigt, dass die Sortierung der Anfragepläne bereits durch das einfache Kostenmodell ohne zusätzliche Berücksichtigung der Sleep-Kosten korrekt vorgenommen wird. Der Sleep-Mode als der Zustand mit dem geringsten Energieverbrauch hat zwar entscheidende Auswirkung auf den Gesamtenergieverbrauch, aber keinen Einfluss auf die relative Ordnung. Somit ist die vorgestellte Kostenabschätzung für die Planbewertung ausreichend.

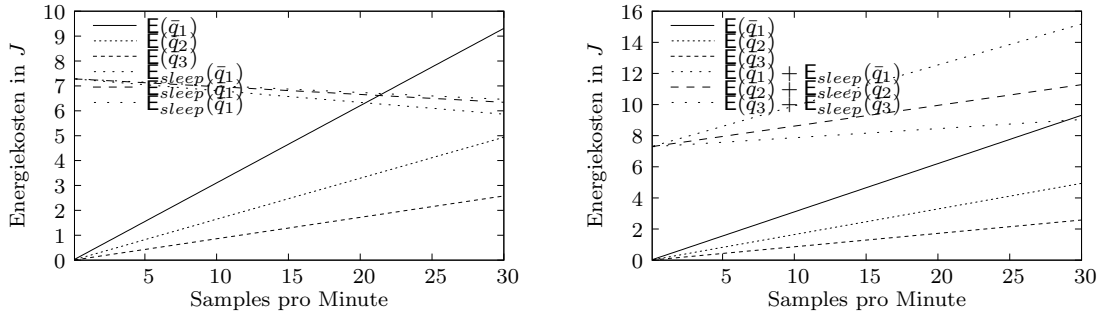
Die Werte zeigen weiterhin, dass beispielsweise bei der Ausführung von Plan $\bar{q}_3$ gegenüber einer Ausführung von Plan $\bar{q}_1$ eine Kosteneinsparung von ca. 8% erreicht werden kann. Mit der gesparten Energie kann das Netzwerk entsprechend länger betrieben werden. D.h. ohne Anpassungen an der Hardware vorzunehmen oder die Datenqualität zum Beispiel durch die Wahl größerer Sampling-Intervalle abzuschwächen, wurde eine achtprozentige Laufzeitverlängerung erzielt.

In Bezug auf das Beispiel fällt auf, dass existierende Lösungen aufgrund des zumeist stark eingeschränkten Funktionsumfangs lediglich eine Teilmenge der von *AnduIN* vorgeschlagenen alternativen Anfragepläne umsetzen können. Konkret bedeutet dies, dass zum Beispiel bei HiFi, welches TinyDB als INQP verwendet, lediglich der teuerste Plan $\bar{q}_1$ realisiert werden könnte. Eine Auslagerung der anderen Operatoren in das WSN ist aufgrund des Fehlens der notwendigen Implementierungen im monolithischen Kern des INQP nicht möglich.

Das vorangegangene Beispiel hat die prinzipielle Eignung des vorgestellten Kostenmodells zur Kostenapproximation gezeigt. Es wurde aber auch deutlich, dass die Sleep-Phase zu den wesentlichen Kostenfaktoren eines WSN zählt. Im Weiteren soll der Einfluss dieser Phase auf den Gesamtenergieverbrauch eines WSN gezeigt werden.

These 5 *Die Sampling-Periode (und die daraus resultierenden Phasen von Inaktivität eines Sensorknotens) hat einen signifikanten Einfluss auf den Energieverbrauch eines WSN.*

Als Beispiel werden die eben evaluierten physischen Anfragepläne $\bar{q}_1, \bar{q}_2$ mit $\bar{q}_3$ mit verschiedenen Sampling-Perioden herangezogen. Es wird ein WSN mit 30 Knoten und einer mittleren Höhe von $h = 3.27$ angenommen (dies entspricht einem vollständigen Binärbaum der Höhe 3). Abbildung 6.8(a) zeigt die (geschätzten) Kosten für das WSN bei steigender Sampling-Periode. Wie zu erwarten war, sinken die Kosten für den Sleep-Mode mit steigender Sampling-Häufigkeit. Es wird auch deutlich, dass die Kosten für



(a) (b)

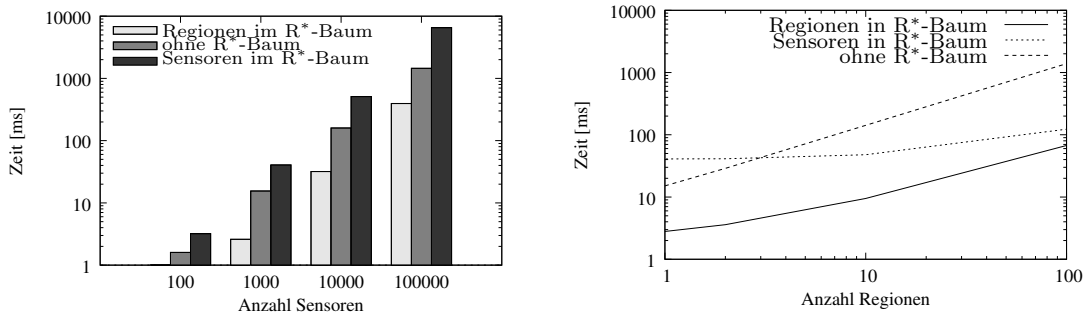
Abbildung 6.8: Energieverbrauch bei variablen Sampling-Raten: (a) Energieverbräuche für Sleep- und Aktiv-Phase im Vergleich (b) Energieverbrauch für Aktiv-Phasen und Gesamtenergieverbrauch im Vergleich

die beiden Anfragepläne $\bar{q}_2$ und $\bar{q}_3$ mit steigender Anzahl von Sensoren und sinkender Sampling-Periode stärker voneinander abweichen. Der kleine Unterschied, welcher sich bei Verwendung eines einzelnen Knotens feststellen lässt, wird durch die höhere Ausführungshäufigkeit entsprechend verstärkt. Die Relationen zueinander bleiben dabei aber erhalten.

Bereits im vorhergehenden Beispiel mit nur einem Sensorknoten machten die Kosten für den Sleep-Mode den größten Kostenanteil aus. Auch in diesem Beispiel mit vielen Knoten sind die Kosten für die Sleep-Phase dominant. Senkt man die Sampling-Periode so weit, dass lediglich einmal pro Minute gemessen wird, so sind die Kosten für das Bereitstellen der Daten, für notwendige Berechnungen und das Versenden vernachlässigbar gering. Mit steigender Sampling-Periode treten die Kosten für die Anfrageverarbeitung stärker in den Vordergrund. Im obigen Beispiel mit einer Sampling-Periode von ca. 3s, also 20 Samples pro Minute, dominieren die Verarbeitungskosten die Gesamtkosten.

Abbildung 6.8(b) zeigt den approximierten Energieverbrauch ohne Kosten für die Sleep-Phase und die Gesamtenergiekosten (aktive Phasen + Sleep-Phasen) im Vergleich. Die Abbildung zeigt sehr gut, dass für eine korrekte Bewertung der durch die Energieverbräuche der einzelnen Anfragepläne bestimmten Reihenfolge die Betrachtung der Kosten für die aktiven Phasen ausreichend ist.

Das Beispiel zeigt, in welchem Bereich das typische Einsatzgebiet von *AnduIN* liegt. Handelt es sich um WSN, welche verhältnismäßig selten Werte erfassen und verarbeiten, so fallen die Vorteile durch die Anfrageoptimierung nicht ins Gewicht. Steigt hingegen der Berechnungsaufwand, zum Beispiel durch eine verhältnismäßig hohe Aktivierungsrate oder durch „teure“ Operatoren, dann bringt die Verwendung von *AnduIN* deutliche Vorteile, was die zu erwartende Lebenszeit des WSN betrifft. Diese Ergebnisse decken sich sehr gut mit den zu Beginn der Arbeit präsentierten Einsatzszenarien.



(a) Eine Region (b) 1000 Sensoren

Abbildung 6.9: Berechnungszeit für die Erstellung der Whitelist in Abhängigkeit von der Anzahl der Sensorknoten für den Inside-Operator

6.2.2 Operatoren

Im folgenden Abschnitt sollen die im Kapitel 5 beschriebenen Operatoren für räumliche Anfragen, das Verfahren für die adaptive Burst-Erkennung und der Ansatz zum quantitativen Frequent Pattern Mining in Datenströmen evaluiert werden.

6.2.2.1 Räumliche Operatoren

Die korrekte Funktion der implementierten räumlichen Operatoren wurde über Modul- und Systemtests geprüft. Die Performance der räumlichen Anfrageverarbeitung wurde im Kontext der DBMS bereits hinreichend evaluiert. Im Folgenden sollen daher lediglich grundlegende Eigenschaften der implementierten Datenstromoperatoren aufgezeigt werden. Zunächst werden hierzu kurz Implementierungsdetails beschrieben. Anschließend folgt eine Betrachtung der initialen Verarbeitungsphase im Fall von sich nicht bewegenden Sensoren, gefolgt von einer Betrachtung der Datenstromanalyse.

Implementierungsdetails Die Implementierung des Inside-Operators setzt unter anderem auf einem R*-Baum auf. Dieser wurde mittels der *SpatialIndex*[6]-Bibliothek realisiert. Für komplexere Operationen auf räumlichen Objekten wie zum Beispiel den Schnitt oder Verbund wurde die *Generic Geometry Library*[7] (GGL) eingesetzt. Die Bestimmung, ob ein Punkt in einer definierten Region liegt, erfolgt anhand des *Winding-Number*-Algorithmus. Dieser testet, wie viele Kanten der gegebenen Region von einem von dem zu prüfenden Punkt ausgehenden Strahl geschnitten werden. Ist die Anzahl der geschnittenen Kanten ungerade, so befindet sich der Punkt innerhalb der Region.

[6]http://trac.gispython.org/spatialindex
[7]http://geometrylibrary.geodan.nl/

Initialisierungsphase Wurden die Koordinaten der Sensoren vorab bei *AnduIN* registriert, dann baut das System zum Zeitpunkt der Anfrage-Initialisierung einmalig eine Whitelist für die spätere Filterung auf. In einem datenstromverarbeitenden System spielt die Initialisierung von Anfragen eine eher untergeordnete Rolle. Um die Gesamtsystemlast möglichst gering zu halten, sollte diese Phase dennoch möglichst effizient sein. Aus diesem Grund kommt zum Beispiel im Fall des Inside-Operators bei der Whitelist-Erstellung ein R*-Baum zum Einsatz. Prinzipiell kommen für die Erstellung der Whitelist drei unterschiedliche Ansätze in Frage:

- Die Regionen werden im R*-Baum verwaltet. Die Sensorknoten dienen als Anfrageobjekte.
- Die Sensorknoten werden im R*-Baum verwaltet. Die Regionen dienen als Anfrageobjekte.
- Es wird auf die Verwendung einer Indexstruktur verzichtet.

Im Weiteren soll folgende These betrachtet werden:

These 6 *Die Performance während der Initialisierung des Inside-Operators hängt stark von der Art der Verwendung der Indexstruktur ab.*

Für die Untersuchung wurden zufällige Polygone mit vier Eckpunkten erzeugt und als solche bei *AnduIN* registriert (nicht als Rechtecke), so dass der Winding-Number-Algorithmus für den Aufbau der Whitlist Verwendung fand.

Abbildung 6.9(a) zeigt die benötigte Zeit für die Bestimmung der Whitelist in Abhängigkeit von der Anzahl der im System registrierten Sensoren. Die Anzahl der Regionen war auf 1 fixiert. Wie zu erwarten war, ist der Ansatz, bei dem im R*-Baum Regionen verwaltet werden, den anderen beiden Lösungen überlegen. Die Kosten steigen dabei linear mit der Anzahl der zu prüfenden Sensoren. In Abbildung 6.9(b) ist die Kostenentwicklung für unterschiedliche Anzahlen von Regionen dargestellt. Die Anzahl der Sensoren war auf 1000 fixiert. Für eine kleine Menge von Anfrage-Regionen ist der Ansatz, bei dem der R*-Baum Regionen verwaltet, die effizienteste Lösung. Die Kosten für diesen Ansatz steigen dabei mit zunehmender Anzahl der Regionen stärker als für den Ansatz, bei dem im R*-Baum Sensorknoten verwaltet werden. Für eine große Anzahl von Regionen ist daher der letztere Ansatz effizienter.

Die Ergebnisse zeigen, dass die Verwendung eines R*-Baumes erwartungsgemäß zu Performancevorteilen führt. Je nach verwendeter Indexvariante („Regionen im R*-Baum“ bzw. „Sensoren im R*-Baum“) und Anzahl der Regionen und Sensoren fällt dieser Vorteil mehr oder weniger stark aus. Interessant ist, dass bei Verwendung von „Regionen im R*-Baum“ die Kosten deutlich schneller steigen als im Fall der „Sensoren im R*-Baum“. Die Ursache hierfür liegt vermutlich in dem großen Aufwand für das Erzeugen der Index-Struktur beim „Regionen im R*-Baum“-Ansatz. Das Einfügen und Suchen scheint hier deutlich aufwändiger zu sein als bei der Verwendung einfacher Sensorknoten. Um diese Vermutung zu überprüfen, sollen im Folgenden daher die Kosten für die

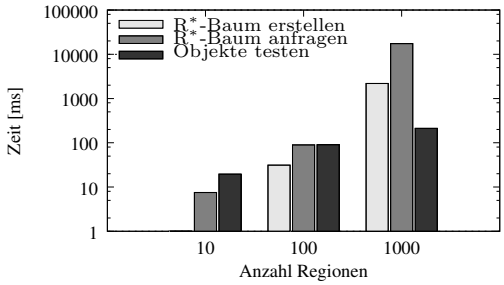


Abbildung 6.10: Laufzeitanalyse für die Whitelist-Erstellung beim Inside-Operator

einzelnen Phasen (R*-Baum erstellen, R*-Baum anfragen und Objekte testen) untersucht werden.

These 7 *Das Erstellen der Indexstruktur ist für einen wesentlichen Teil der Kosten bei der Initialisierung des Inside-Operators verantwortlich.*

Abbildung 6.10 zeigt die anfallenden Kosten für die unterschiedlichen Phasen der Initialisierung. Im Beispiel wurden die Anfrage-Regionen im R*-Baum gespeichert. Die Abbildung zeigt sehr gut, wie der Anteil der durch die Anfragebaum-Erstellung entstehenden Kosten mit der Anzahl der Anfrageregionen kontinuierlich steigt und im Fall von 1000 Anfrageregionen deutlich mehr Kosten verursacht als der finale Vergleich. Aufgrund der Prüfung mittels der Indexstruktur werden im Beispiel in allen drei Fällen ca. 85% der Regionen direkt verworfen. Lediglich 15% der Regionen müssen abschließend mittels des Winding-Number-Verfahrens im Detail geprüft werden. Die Grafik zeigt sehr gut, dass die Indexerstellung für den starken Anstieg der Kosten in Abbildung 6.9(b) verantwortlich ist.

Die obigen Ergebnisse haben für den Inside-Operator gezeigt, dass der native Ansatz „Regionen im R*-Baum“ nicht in jedem Fall die beste Lösung darstellt. Es sollte immer in Abhängigkeit von der Anzahl der zu prüfenden Sensoren und der Anzahl der Regionen entschieden werden, über welchen Objekten die Indexstruktur aufgebaut wird. Weiterhin wurde gezeigt, dass mit steigender Objektanzahl im R*-Baum der Aufwand für die Indexstruktur-Erzeugung so stark steigt, dass die Kosten für das Überprüfen eines Objektes letztendlich nur noch einen geringen Teil der Kosten ausmachen. An dieser Stelle wäre daher zu überlegen, ob nicht die Sensorknoten dauerhaft in einem systemweit verwendeten R*-Baum verwaltet werden sollten. Alle räumlichen Anfragen könnten mittels dieser Indexstruktur beantwortet werden. Damit könnten wiederholt anfallende Konstruktionskosten eingespart werden.

Verarbeitungsphase Bei allen in der Arbeit vorgestellten räumlichen Operatoren handelt es sich im Wesentlichen um Filteroperatoren. Im Weiteren soll daher nachstehende These verifiziert werden:

These 8 *Die in* AnduIN *integrierten räumlichen Operatoren benötigen für die Verarbeitung eines Datensatzes nur unwesentlich mehr Zeit als der einfache Filteroperator.*

Operator	Tupelrate (in $Tupel/s$)	Zeit pro Tupel (in μs)	
		sphärisch	kartesisch
einfacher Filter	7352.0	136.0	-
Inside-Operator	5995.6	166.8	-
Nearest-Neighbor-Operator	5156.7	193.92	190.7
Within-Distance-Operator	5377.5	185.96	157.49

Tabelle 6.4: Tupelraten und Verarbeitungszeit für verschiedene räumliche Operatoren über Daten von mobilen Sensorknoten

Aufgrund des höheren Aufwandes soll sich im Weiteren auf mobile Sensoren beschränkt werden. Bei diesen ist zunächst die aktuelle Position der Sensoren aus dem Datensatz herauszulösen und anschließend je nach Operator zu analysieren. Im Fall des Inside-Operators muss hierzu zunächst eine Anfrage an den R*-Baum gestellt werden. Wird eine MBB gefunden, ist eine detaillierte Prüfung durchzuführen.

In Tabelle 6.4 sind für die implementierten Operatoren die Verarbeitungszeiten pro Tupel und der daraus resultierende Datendurchsatz dargestellt. Die ersten Tests wurden dabei mit sphärischen Koordinaten durchgeführt. Erwartungsgemäß weisen alle drei Verfahren einen geringeren Durchsatz als der einfache Filter auf. Dennoch sind alle Implementierungen deutlich schneller als beispielsweise eine Aggregation oder ein Verbund (siehe Tabelle 6.2). Was weiterhin auffällt, ist der geringere Datendurchsatz bei den beiden abstandsbasierten Verfahren. Da die ersten Tests unter Verwendung sphärischer Koordinaten (GPS-Koordinaten) erfolgten, war die Abstandsberechnung entsprechend komplex (im Fall des kartesischen Bezugssystems erfolgte die Berechnung auf Grundlage des euklidischen Abstands, beim sphärischen Bezugssystem wurde die Abstandsberechnung nach [208] vorgenommen). Daher wurde für die beiden Verfahren exemplarisch eine Messung unter Verwendung kartesischer Koordindaten durchgeführt. Tabelle 6.4 (rechte Spalte) zeigt den verhältnismäßig starken Einfluss des gewählten Koordinatensystems. Vor allem der Within-Distanze-Operator wird beim Einsatz kartesischer Daten deutlich leistungsfähiger.

Die Ergebnisse zeigen, dass die Verarbeitung räumlicher Operatoren in *AnduIN* auch bei großen Datenmengen effizient möglich ist. Die Kosten bewegen sich deutlich unterhalb der Kosten für komplexe Operatoren. Weiterhin wurde ein verhältnismäßig starker Einfluss des verwendeten Koordinatensystems ausgemacht. Prinzipiell sollte dies kein Problem bei der Verarbeitung großer Datenmengen darstellen, solange deren Verarbeitung auf der zentralen Komponente von *AnduIN* stattfindet. Überträgt man die Mehrkosten aber auf die In-Network-Analyse, dann ergeben sich hieraus auch höhere Energieverbräuche, und es ist abzuwägen, ob die Mehrkosten durch den höheren Berechnungsaufwand gegenüber der Positionsangabe durch die sphärischen Koordinaten gerechtfertigt sind.

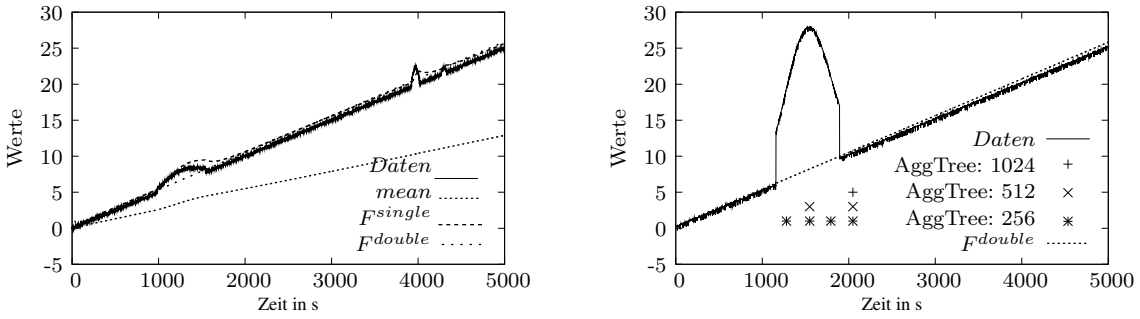


Abbildung 6.11: Burst-Erkennung für trendbehaftete Daten

6.2.2.2 Burst-Erkennung

Im Folgenden soll das in Abschnitt 5.2 vorgestellte Verfahren zur adaptiven Burst-Detektion untersucht werden. Zunächst soll geprüft werden, ob und inwieweit durch das vorgeschlagene Verfahren Bursts in trendbehafteten Daten besser erkannt werden. Hierzu wird das Verfahren mit dem ursprünglich von Zhang [228] entwickelten Verfahren, auf welchem dieses basiert, verglichen. Weiterhin soll untersucht werden, ob es einen Einfluss der verwendeten Parameter auf die Qualität des vorgestellten Verfahrens gibt und wenn ja, wie sieht dieser aus?

These 9 *Das in Abschnitt 5.2 vorgestellte Verfahren zur adaptiven Burst-Erkennung ist für trendbehaftete Daten besser geeignet als der in [229] präsentierte Ansatz.*

Abbildung 6.11(a) zeigt einen Datensatz, welcher sowohl einen starken Trend als auch Bursts aufweist. Für den Datensatz wurden zudem der bei Zhang verwendete Mittelwert (*mean*) sowie die beim Verfahren der adaptiven Burst-Erkennung verwendete Vorhersagen (F^{single} und F^{double}) abgebildet. Die Abbildung verdeutlicht den wesentlichen Schwachpunkt des originalen mittelwertbasierten Ansatzes. Das Verfahren nach Zhang würde fälschlicherweise den kompletten Datensatz als Ausreißer erkennen. Zugleich zeigt die Abbildung die Funktion des präsentierten Ansatzes. F^{double} (mit $\alpha = 0.1, \gamma = 0.25$ und 500) zeigt den Verlauf der approximierten Werte bei Verwendung der doppelten exponentiellen Glättung. Im Gegensatz zum Originalansatz heben sich bei der Verwendung der doppelten exponentiellen Glättung die Bursts deutlich von dem zugrunde liegenden Datensatz ab.

Anders verhält sich dies bei der einfachen exponentiellen Glättung (F^{single} mit $\alpha = 0.01$), welche im Wesentlichen eine Ein-Schritt-Vorhersage macht. Das Verfahren folgt dem tatsächlichen Verlauf so stark, dass eine Burst-Erkennung kaum möglich ist.

In Abbildung 6.11(b) wird die Menge der vom Operator erzeugten Burst-Nachrichten gezeigt. Für die Überwachung wurden Aggregationsbäume mit 256, 512 bzw. 1024 Elementen auf Ebene 0 verwendet. Die unterhalb des Bursts eingezeichneten Punkte zeigen den Zeitpunkt der Benachrichtigung des Nutzers über einen eingetretenen Burst an. Der

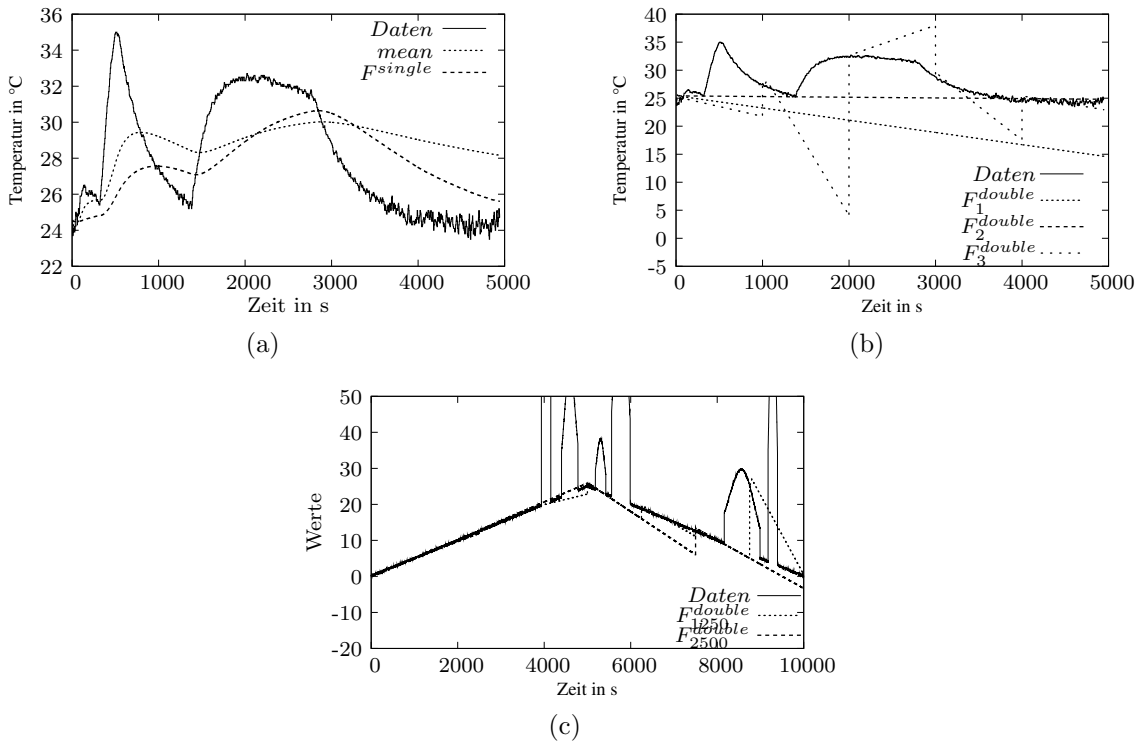


Abbildung 6.12: Einfluss verschiedener Parameter auf die Güte der Burst-Erkennung

Burst wurde in allen drei Varianten korrekt erkannt und gemeldet, wobei je nach Fenstergröße eine bis vier Nachrichten verursacht werden. Obwohl die Vorhersage in diesem Beispiel nur auf einer Vorhersageweite von 50 Zeitpunkten basierte, wurde der Burst richtig erkannt. Die Ursache hierfür liegt in dem Verzicht auf eine erneute Initialisierung der Vorhersage innerhalb des Ereignisses. Der vor Beginn des Bursts bestimmte Trend wurde bis zu dessen Ende weiter verfolgt.

Diese ersten Ergebnisse zeigen den Mehrwert des vorgestellten Verfahrens gegenüber dem ursprünglich von Zhang vorgeschlagenen Ansatz. Im Weiteren soll nun der Einfluss der verschiedenen Parameter auf die Güte der Bursterkennung untersucht werden. Es wird folgende These geprüft:

These 10 *Die Güte der adaptiven Burst-Erkennung ist stark von den Parametern des Vorhersageverfahrens abhängig.*

Zunächst soll der Einfluss der Glättungsparameter auf das Verfahren untersucht werden. Hierzu wurden Temperaturmessungen auf einem Sensorknoten durchgeführt. Durch die externe Zufuhr von Wärme wurden zwei Bursts erzeugt. Der erste Burst kam dabei durch das schnelle Hinzufügen und Entfernen der Wärmequelle zustande. Für den zweiten Burst wurde die Wärmequelle über längere Zeit in der Nähe der Sensorknoten aufgestellt, was zu einer langsameren Abkühlung führte.

In Abbildung 6.12(a) wird der ursprüngliche Mittelwert-Ansatz *mean* von Zhang mit

der einfachen exponentiellen Glättung F^{single} ($\alpha = 0.01$) verglichen. Beide Ansätze folgen den Daten mehr oder weniger genau. Im Fall des Mittelwertes hat die Intensität des Bursts einen stärkeren Einfluss als bei der einfachen Glättung. Nichtsdestotrotz führen beide Ansätze zur korrekten Erkennung der Bursts, wobei die einfache Glättung diese früher und auch von der Dauer besser erkennt.

Abbildung 6.12(b) zeigt das gleiche Beispiel, wobei die Burst-Überwachung auf Basis der doppelten exponentiellen Glättung F^{double} durchgeführt wurde. Es wurden drei verschiedene Glättungsparameter untersucht: F_1^{double} mit $\alpha = 0.01, \gamma = 0.01$, F_2^{double} mit $\alpha = 0.01, \gamma = 0.16$ und einer maximalen Vorhersageweite von 50 Schritten und F_3^{double} mit $\alpha = 0.01, \gamma = 0.16$ und einer 1000-Schritt-Vorhersage. Eine detaillierte Beschreibung der Parameter findet sich im Anhang.

In Abbildung 6.12(b) wird das wesentliche Problem des präsentierten Ansatzes deutlich. Das vorgestellte Verfahren hängt stark von der Wahl der Glättungsparameter und dem Fenster für die Vorhersage ab. Im Fall von F_2^{double} wird die Kurve sehr gut approximiert, wohingegen die anderen Approximationen zu schlechten Ergebnissen führen.

These 11 *Das vorgestellte Verfahren der adaptiven Burst-Erkennung passt sich gut an plötzliche Änderungen im Trend an.*

Um diese These zu überprüfen, wurde wiederum ein künstlicher Datensatz erzeugt, welcher zunächst einen stark positiven Trend und anschließend einen stark negativen Trend aufweist. Zusätzlich wurde der Datensatz wiederum mit Bursts überlagert.

Für die Burst-Erkennung wurden zwei unterschiedliche Vorhersagefenster (1250 und 2500 Schritte) verwendet. Abbildung 6.12(c) zeigt den Datensatz und die approximierten Werte $F_1^{double}250$ und $F_2^{double}500$ mit den Parametern $\alpha = 0.01, \gamma = 0.135$. Beide Verfahren approximieren den Trend weitestgehend korrekt. Das Verfahren mit der größeren Vorhersageweite passt sich erwartungsgemäß langsamer an die Trendänderung an, als dies bei dem Verfahren mit der kleineren Vorhersageweite der Fall ist. Der vorletzte Burst wird von dem Verfahren mit der kürzeren Vorhersageweite nicht bis zum Ende korrekt erkannt. Die Folge ist, dass die neuerliche Approximation innerhalb des Bursts beginnt. Die Ursache hierfür liegt in der zu klein gewählten Fensterweite.

These 12 *Eine Burst-Erkennung über stark schwankenden Datenwerten ist weder mit dem Verfahren nach [229] noch mit dem präsentierten Verfahren der adaptiven Burst-Erkennung möglich.*

Es sollen im Weiteren Kursdaten herangezogen werden, welche oftmals zwar einen generellen Trend aufweisen, häufig aber auch starken lokalen Schwankungen unterliegen. Als Beispiel-Datensatz diente der Wechselkurs Schweizer Franken in US Dollar aus den Jahren 1985-1991[8].

In Abbildung 6.13(a) werden die für statische Daten geeigneten Burst-Erkennungsverfahren *mean* und F^{single} ($\alpha = 0.0005$) gezeigt. Die Abweichung der approximierten Werte vom eigentlichen Datensatz ist deutlich zu erkennen. Entsprechend schlechte Ergebnisse würde eine Burst-Erkennung liefern. Abbildung 6.13(b) zeigt einen Ausschnitt

[8] http://www.cs.ucr.edu/∼eamonn/iSAX/iSAX.html

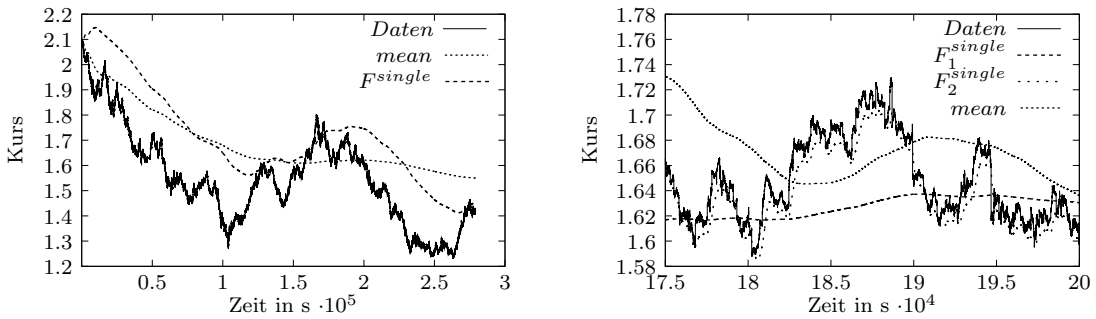


Abbildung 6.13: Burst-Erkennung über stark schwankenden Datenwerten

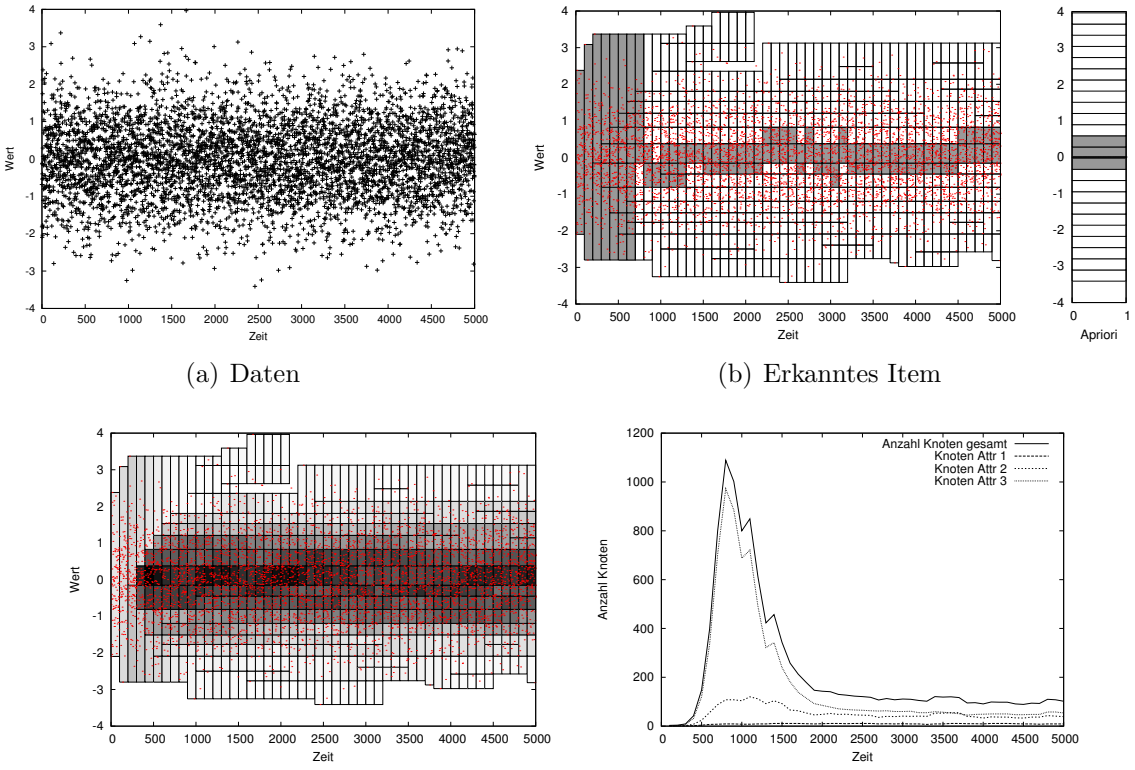
der Daten aus Abbildung 6.13(a). Hier wurde neben der einfachen Glättung F_1^{single} mit $\alpha = 0.0005, w = 1000$ und F_1^{single} mit $\alpha = 0.00005, w = 100000$) zusätzlich der Mittelwertansatz *mean* mit einer Fenstergröße von 1000 untersucht. Die von diesem Verfahren produzierte Approximation passt sich der Datenkurve sehr gut an, was allerdings eine zu feingranulare Burst-Erkennung zur Folge hat.

Aufgrund des Fehlens eines generellen Trends ist die doppelte exponentielle Glättung für den vorliegende Datensatz schlecht geeignet. Aufgrund des hohen Anteils an Trendänderungen müssten die Parameter kontinuierlich angepasst werden. Eine Ein-Schritt-Vorhersage würde dies leisten, entspricht aber der einfachen exponentiellen Glättung. Da diese sich dem Trend aber zu stark anpasst, ist eine Burst-Erkennung über stark schwankenden Daten nicht möglich.

Der zurückliegende Abschnitt hat gezeigt, dass das präsentierte Verfahren der adaptiven Burst-Erkennung für nichtstationäre Daten bessere Ergebnisse als der ursprüngliche mittelwertbasierte Ansatz liefert. Die Ergebnisse haben allerdings auch Defizite des vorgestellten Verfahrens aufgezeigt. So hängt die Qualität der Burst-Erkennung im Fall der doppelten exponentiellen Glättung stark von den verwendeten Parametern für das Vorhersageverfahren und der Vorhersagewerte ab. Des Weiteren wurde deutlich, dass die Eigenschaften der Daten ebenfalls einen starken Einfluss auf das Resultat haben, so zeigen sich die Vorteile des präsentierten Verfahrens überwiegend bei trendbehafteten Daten. Sind die Daten stationär oder lässt sich kein eindeutiger Trend identifizieren, so sind andere Lösungsansätze zu präferieren. Das einzusetzende Verfahren sollte daher in Abhängigkeit von den zu erwartenden Daten gewählt werden.

6.2.2.3 Quantitatives Frequent Pattern Mining

In diesem Abschnitt soll das entwickelte Verfahren zum Quantitativen Frequent Pattern Mining analysiert werden. Es werden dabei insbesondere die Güte der gelieferten Ergebnisse und der Einfluss der verschiedenen Parameter des Verfahrens untersucht. Zuletzt soll noch auf den Energieverbrauch der In-Network-Komponenten der entsprechenden



(a) Daten

(b) Erkanntes Item

(c) Informationsdichte

(d) Knotenanzahl im FP^2-Stream

Abbildung 6.14: Standardnormalverteilte Daten

Operatoren eingegangen werden.

These 13 *Das in Abschnitt 5.3 präsentierte FP^2-Stream-Verfahren ist für das Auffinden häufiger Muster in Sensordatenströmen mit quantitativen Attributen geeignet.*

Hierzu wurden drei Datenströme zu je 5000 Datenpunkten (aus $\mathbb{R}$) erzeugt. Ein Datenstrom lieferte Werte über einem Attribut, d.h., die Analyse erfolgte im Weiteren über insgesamt drei Attributen. Die erzeugten Werte waren standardnormalverteilt, was weitestgehend dem Verhalten von realen Sensordaten entspricht (je nach Genauigkeit der eingesetzten Technik schwanken die erfassten Werte mehr oder weniger stark um den realen Wert). Abbildung 6.14(a) zeigt exemplarisch den erzeugten Datenstrom für ein Attribut.

Aus diesen drei Datenströmen sollten nun häufige Itemsets extrahiert werden, wobei pro Attribut (Datenstrom) ein Item gefunden werden sollte. Aufgrund der Wahrscheinlichkeitsdichte der Standardnormalverteilung ist zu erwarten, dass sich die Items und damit auch die Itemsets um den Mittelwert der Datenverteilung herum aufbauen.

Im ersten Test sollten Items bzw. Itemsets mit einem minimalen Support von $minsup = 0.25$ extrahiert werden. Es wurde mit einer Batchgröße von $|B| = 100$, einer Fenstergröße von $w = 5$, Histogrammen mit $d = 10$ Buckets und einer maximalen Schiefe von

$maxskew = 0.2$ getestet. Abbildung 6.14(b) zeigt beispielhaft die zeitliche Entwicklung des über dem Datenstrom aus Abbildung 6.14(a) extrahierten Items. Eine vertikale Reihe Boxen entspricht dabei den Knoten im FP^2-Stream, die über diesem Attribut zum gegebenen Zeitpunkt existieren. Weiße Boxen repräsentieren Knoten welche nicht häufig sind. Graue Balken, die sich über mehrere Boxen erstrecken können, stellen Items dar, die im FP^2-Stream über mehrere Knoten verteilt sind; pro Zeitschritt wird jeweils ein solches extrahiert.

Zum Vergleich ist in Abbildung 6.14(b) (rechts) das Item dargestellt, welches über demselben Attribut durch den Apriori-Ansatz von Srikant nach dem Durchlauf des kompletten Datensatzes erzeugt wird. Der direkte Vergleich zeigt, dass das vorgestellte Verfahren das Item in den nahezu gleichen Grenzen findet. Die Schwankungen beim FP^2-Stream kommen durch das Rauschen der Daten zustande. Aufgrund der Betrachtung einer viel kürzeren Vergangenheit kommen die Schwankungen entsprechend stärker zum Tragen. Im Weiteren soll genauer auf die Eigenschaften des FP^2-Stream-Verfahrens eingegangen werden.

In Abbildung 6.14(b) ist die Initialisierungsphase des Verfahrens sehr gut zu erkennen. Zu Beginn der Analyse wird der Wertebereich durch das Item vollständig überdeckt. Anschließend wird der alles überdeckende Knoten in mehrere Teilknoten zerlegt. Aufgrund des Zeitfensters von fünf Batches setzt sich diese Verfeinerung allerdings erst nach 600 Zeiteinheiten durch. Anschließend schwankt das erzeugte Item um den Mittelwert der standardnormalverteilten Daten.

Interessant ist das beobachtete Verhalten im Bereich zwischen den Zeitpunkten 1500 und 2000 und oberhalb des Wertes 2.5. Hier lösen sich Knoten vom Basisintervall ab, da über längere Zeit offensichtlich keine Werte in diesem Bereich eingetroffen sind. Aufgrund des Fehlens von neu eintreffenden Werten in diesem abgelösten Bereich wird er zum Zeitpunkt 2200 aus dem FP^2-Stream entfernt.

Elementar für die Effektivität und die Effizienz des Verfahrens ist das verwendete Maß der Informationsdichte. Abbildung 6.14(c) zeigt für den betrachteten Beispieldatenstrom die Dichte für die einzelnen Items im FP^2-Stream. Die Itemgrenzen entsprechen denen aus Abbildung 6.14(b). Die Abbildung zeigt sehr gut, wie die Informationsdichte mit zunehmender Knotenverfeinerung steigt, und dass sie erwartungsgemäß um den Mittelwert am größten ist. Im Fall, dass keines der Items den geforderten minimalen Support aufweist, startet die Rekombination während der FP^2-Growth-Phase am Item mit der höchsten Informationsdichte.

Der FP^2-Stream ist eine Datenstruktur zur effizienten Speicherung von Transaktionen. Das kontinuierliche Verfeinern von Knoten hat allerdings einen starken Einfluss auf die Größe der Baumstruktur. So führt zum Beispiel eine Verfeinerung der Wurzel zu einer Verdoppelung aller abhängigen Knoten. Während der initialen Phase muss sich das Verfahren erst an die Daten anpassen. Dies kann eine Vielzahl von Split- und Verschmelzungsoperationen zur Folge haben. In Abbildung 6.14(d) ist die zeitliche Entwicklung der Knotenanzahl zu obigem Beispiel dargestellt. Sehr gut zu erkennen ist die Spitze zu Beginn des Verarbeitungsprozesses. An dieser Stelle übersteigt die Knotenanzahl

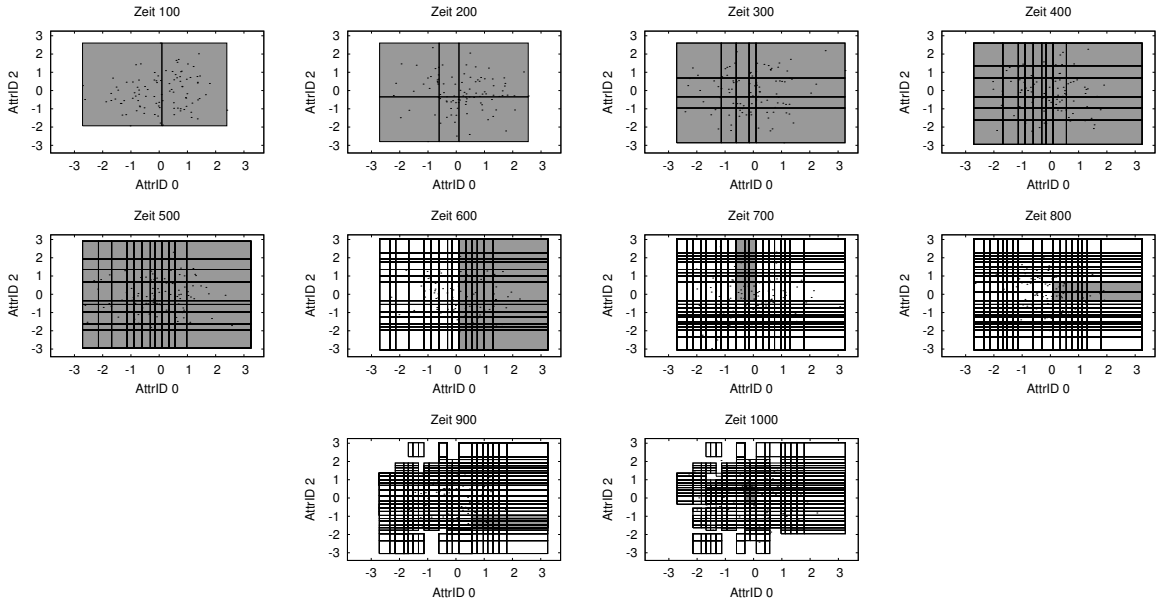


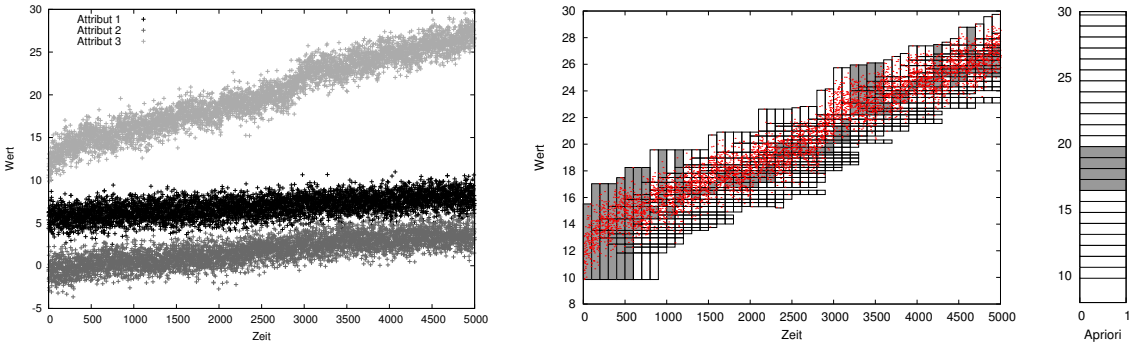
Abbildung 6.15: Entwicklung des 2-elementigen Itemsets über den Attributen 0 und 2

sogar die Anzahl der Transaktionen (500 Transaktionen) im FP^2-Stream. Nach dieser initialen Phase fällt die Anzahl der Knoten im Baum auf ca. 100-150. Da sich keine wesentlichen Änderungen in der Datenverteilung ergeben, bleibt die Knotenanzahl auf diesem Level in etwa konstant.

Abbildung 6.14(d) zeigt außerdem die Anzahl der Knoten pro Attribut im Baum. Erwartungsgemäß steigt diese Zahl mit der Tiefe des Baumes. Das Attribut, welches sich auf Blattebene befindet (im Beispiel Attribut 3), wird also durch die höchste Anzahl an Knoten repräsentiert.

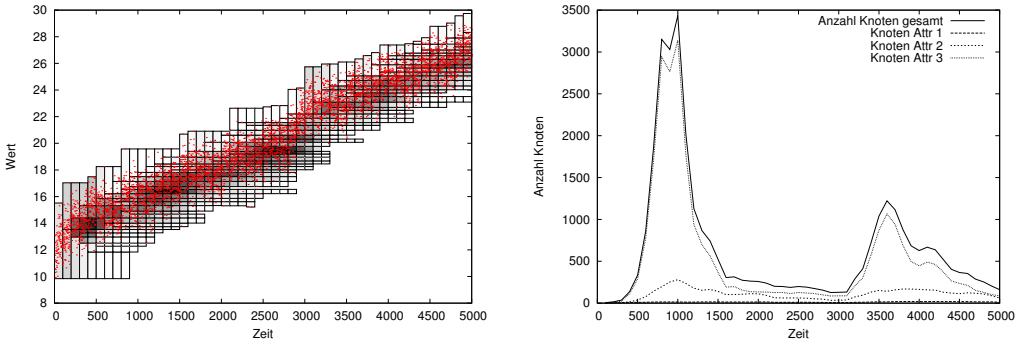
Durch das Rekombinieren von Items innerhalb des FP^2-Stream werden automatisch auch die entsprechenden Itemsets zusammengeführt. Während der Itemset-Exktraktion werden alle Items aus der Datenstruktur herausgelöst, welche dem minimalen Support genügen. Im Beispiel sind dies neben den einelementigen Itemsets zusätzlich 2- und vereinzelt auch 3-Itemsets. Die Bilder in Abbildung 6.15 zeigen die zeitliche Entwicklung des 2-elementigen Itemsets über den Attributen 0 und 2. Jedes der einzelnen Bilder entspricht dabei dem am Ende eines Batches extrahierten Itemset. Die einzelnen Boxen entsprechen wieder den Items im FP^2-Stream. Das extrahierte Itemset ist grau hinterlegt.

In den Abbildungen ist die initiale Phase wiederum sehr gut zu erkennen. Zunächst überdeckt das Itemset den vollständigen Wertebereich in beiden Dimensionen. Anschließend werden die beiden Attribute durch Splits verfeinert. Aufgrund der verwendeten Fenstergröße 5 setzen sich diese Anpassungen erst ab dem Zeitpunkt 600 durch. Im Weiteren pendelt sich das Itemset um die Mittelwerte in beiden Dimensionen ein. Sehr gut zu erkennen ist auch, dass Itemsets, die den Rand des Wertebereichs repräsentieren,



(a) Daten

(b) Item Attribut 3



(c) Dichteverteilung

(d) Anzahl Knoten

Abbildung 6.16: Trendbehaftete Daten

infolge zu weniger Werte in diesen Regionen mit der Zeit entfernt werden. Dies ist vermutlich einer der Gründe, warum sich die Anzahl an Knoten im FP^2-Stream nach dem initialen Anstieg auf einem deutlich niedrigeren Level einpendelt.

Diese ersten Ergebnisse zeigen, dass das präsentierte Verfahren für das Ermitteln von häufigen Mustern über quantitativen Attributen in Datenströmen geeignet ist. In einem nächsten Test soll das Verhalten im Fall von trendbehafteten Daten evaluiert werden.

These 14 *Das FP^2-Stream-Verfahren erzielt für trendbehaftete Datenströme bessere Ergebnisse als der in [193] präsentierte Ansatz.*

Für diese Untersuchung wurden ebenfalls drei Datenströme erzeugt. Jeder Datenstrom umfasste dabei 5000 Datenpunkte, deren Werte einen Trend aufwiesen. Zusätzlich wurden die Daten mit normalverteiltem Rauschen überlagert. Abbildung 6.16(a) zeigt die drei Datenströme im zeitlichen Verlauf. Da sich die Mittelwerte aller Attribute kontinuierlich ändern, ist zu erwarten, dass das System die häufigen Items und Itemsets ebenfalls über die Zeit anpasst.

Das Experiment wurde mit $|B| = 100$, $w = 5$ und einer maximalen Schiefe von $maxskew = 0.2$ und 10 Histogrammbuckets durchgeführt. Der geforderte Support betrug 0.15.

Abbildung 6.16(b) zeigt exemplarisch die Entwicklung der Items für das Attribut 3. Im rechten Teil der Abbildung wurde wieder das durch den Apriori-Ansatz von Srikant auf Basis des gesamten Datenstromes bestimmte Item dargestellt. Aufgrund des Fehlens der zeitlichen Komponente bestimmt dieser lediglich ein Item. Im Gegensatz dazu folgt die Entwicklung der Items, welche durch den FP^2-Stream bestimmt wurden, dem Trend. Die Einschwingphase ist wieder deutlich zu erkennen. Interessant ist die Entwicklung des Items um den Zeitpunkt 1000. Hier sorgen die neu eintreffenden Werte offensichtlich für ein Oszillieren zwischen zwei Items, von denen sich schließlich ab dem Zeitpunkt 1300 eines durchsetzt.

Interessant im direkten Vergleich mit dem Apriori-Ansatz ist auch die Entwicklung der Partitionen. Da beim FP^2-Stream die minimalen und maximalen Werte kontinuierlich angepasst werden, verändert sich auch die Größe der einzelnen Basisintervalle. Aufgrund der feinerer Auflösung ist letztendlich auch eine deutlich bessere Approximation der Itemgrenzen möglich.

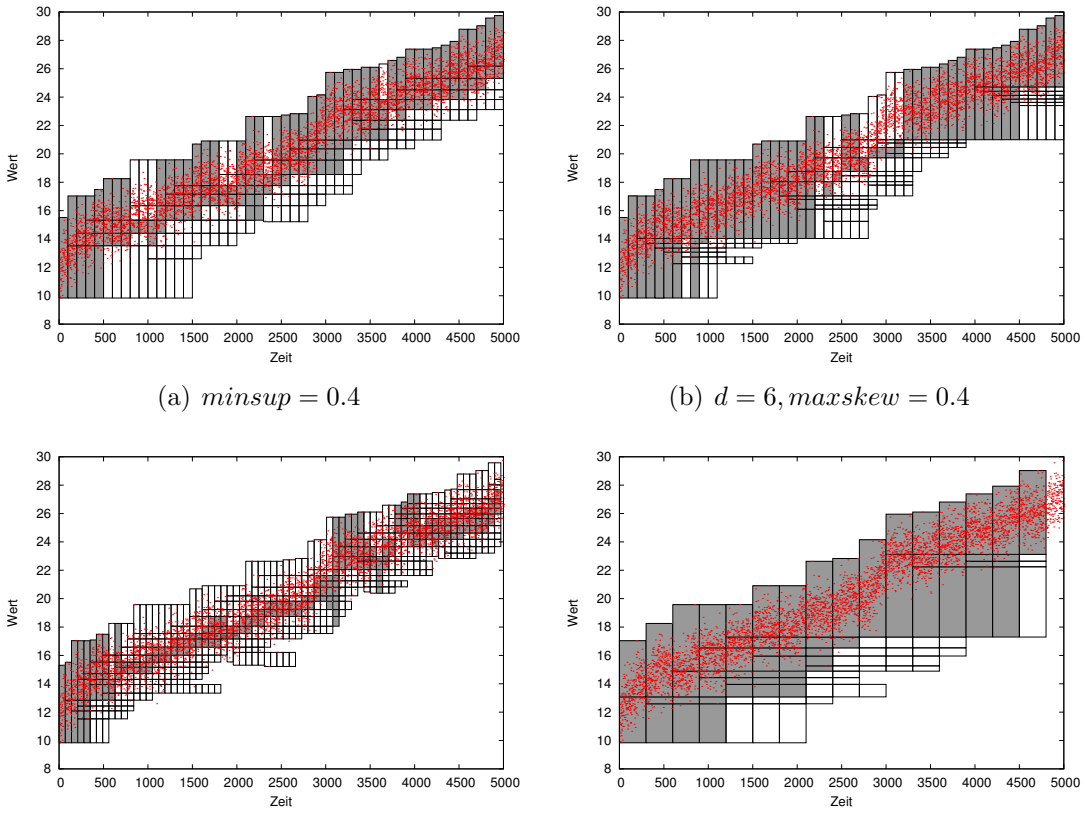
In Abbildung 6.16(c) ist die Informationsdichte-Entwicklung der Items über dem abgebildeten Attribut dargestellt. Abbildung 6.16(d) gibt einen Überblick über die zeitliche Entwicklung der Knotenanzahl im FP^2-Stream. Die Einschwingphase ist hier ebenfalls sehr gut zu erkennen. Trotz des kontinuierlichen Trends pendelt sich die Gesamtknotenanzahl auf einem relativ konstanten Wert bei ca. 200 Knoten ein. Beginnend ab Zeitpunkt 3300 wächst die Knotenanzahl erneut drastisch an. Die Ursache hierfür liegt offensichtlich in einer Änderung der Datencharakteristik während dieser Zeit bei Attribut 3. Trotz der Tatsache, dass es sich um trendbehaftete Daten mit normalverteiltem Rauschen handelt, scheint zwischen den Zeitpunkten 3000 und 3500 eine weitere Überlagerung (ähnlich einem Burst) aufzutreten. Diese plötzliche Veränderung hat entsprechende Auswirkungen auf die Knoten und deren Dichten, welche Attribut 3 repräsentieren (siehe Abbildung 6.16(c)) und führen vermutlich zu dem kurzzeitigen Anstieg der Knotenanzahl.

Im Weiteren soll untersucht werden, inwieweit die verwendeten Parameter Einfluss auf die Item- bzw. Itemset-Entwicklung haben.

These 15 *Die Wahl der Parameter des FP^2-Stream-Verfahrens hat einen starken Einfluss auf die Qualität der erzeugten Itemsets.*

Den weiteren Tests liegen die trendbehafteten Daten aus dem vorhergehenden Experiment zugrunde. Als Basis für den Vergleich der verschiedenen Testläufe mit unterschiedlichen Parameterkonfigurationen dienen die Ergebnisse des letzten Experiments. Alle weiteren Tests unterscheiden sich von diesen in maximal zwei Parametern.

Zunächst soll der Einfluss des minimalen Supports untersucht werden. Für das erste Experiment wurde ein minimaler Support von $minsup = 0.4$ verwendet. Die minimale zu erzeugende Intervallgröße im FP^2-Stream wurde, wie bereits diskutiert, nach [193] bestimmt. Diese ändert sich in Abhängigkeit vom minimalen Support, was sich in Abbildung 6.17(a) durch die größeren Intervalle im Vergleich zu Abbildung 6.16(b) äußert. Erwartungsgemäß überdecken die Items bei einem größeren minimalen Support auch einen größeren Wertebereich. Da die Werte stetig größer werden, erweitern sich die



(a) $minsup = 0.4$

(b) $d = 6, maxskew = 0.4$

(c) $|B| = 70$

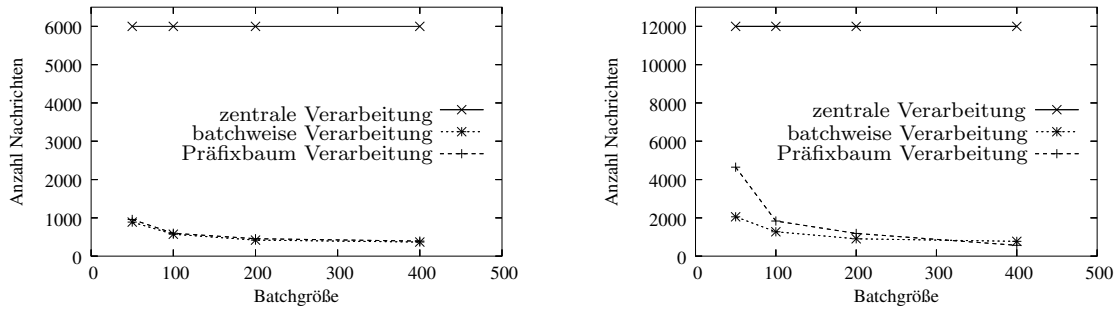
(d) $|B| = 300$

Abbildung 6.17: Frequent Itemset Mining über trendbehafteten Daten: Testläufe mit verschiedenen Parameterkonfigurationen

Items entsprechend mit der Zeit in Richtung der größeren Werte. Diese Werte werden dann erst langsam mit der Zeit verfeinert.

In dem in Abbildung 6.17(b) dargestellten Testlauf wurde der Einfluss des Histogramms auf die Items der Header-Tabelle untersucht. Mit lediglich sechs Buckets wurde die Bucketanzahl gegenüber obigem Beispiel verkleinert, die geforderte Schiefe als Aufspaltungskriterium von Items wurde hingegen vergrößert. Somit kommt es entsprechend seltener zu Verfeinerungen. Eine Dichteveränderung muss sich erst deutlich durchsetzen, ehe es zu einem Split kommt. Dies zeigt sich in einer eher sprunghaften Entwicklung bei den Splits (die Splits erfolgen im Beispiel zu den Zeitpunkten 500. 2000 und 4500).

FP^2-Stream ist ein batchbasiertes Verfahren. Interessant ist daher auch der Einfluss der gewählten Batchgröße. Die Abbildungen 6.17(c) und 6.17(d) zeigen die Analyse für Batchgrößen von 70 bzw. 300. Deutlich zu erkennen ist die sehr gute Item-Extraktion im Fall einer kleineren Batchgröße. Im Gegensatz dazu ist der Trend der Daten so stark, dass die Verfeinerung der Items im Fall von 300 Datensätzen pro Batch nicht ausreichend schnell anschlägt. D.h., das kontinuierliche Splitten hält mit der Entwicklung des



(a) (b)

Abbildung 6.18: Anzahl der versendeten Nachrichten: (a) Sensorknoten mit ID 2, (b) Sensorknoten mit ID 1

Trends nicht mit. Dies resultiert in einer eher schlechten Approximation der Items.

In-Network-Verarbeitung Im Weiteren sollen die drei in Abschnitt 5.3.3 vorgestellten Varianten (vollständig im DSMS durchgeführte Berechnung, batchweise Verarbeitung und Präfixbaum batchweise Verarbeitung) des FP^2-Stream miteinander verglichen werden. Nach den vorangegangenen Untersuchungen stehen hierbei drei wesentliche Aspekte im Vordergrund:

1. der Einfluss der Batchgröße auf die Anzahl der Nachrichten, die versendet werden müssen,
2. der Speicherbedarf auf den Sensoren und
3. der Energieverbrauch auf den Sensoren.

Zunächst soll nachstehende These geprüft werden:

These 16 *Die In-Network-Verarbeitung des FP^2-Stream kann zu einer deutlichen Reduktion der zu versendeten Nachrichten und damit zu einem geringeren Energieverbrauch des WSN führen.*

Die Untersuchungen erfolgten auf Basis der in Abschnitt 6.1.3.2 beschriebenen Simulationsumgebung. Es wurde mit jeweils 3 Sensorknoten gearbeitet, wobei ein Sensorknoten mit der Basisstation verbunden war. Der Sensorknoten auf der Basisstation erfasste Temperaturwerte, die beiden batteriebetriebenen Sensorknoten jeweils Temperatur und Luftfeuchtigkeit. Die Experimente erfolgten mit 6000 Messpunkten pro physikalischem Sensor. In allen Tests waren die Sensorknoten in einer logischen Kette angeordnet. Der Basisknoten bildete dabei den Beginn der Kette und hatte die ID 0. Der zweite Knoten in der Kette hatte die ID 1. Der Knoten mit ID 2 bildete das Kettenende.

In einem ersten Test soll die Anzahl der zu versendenden Nachrichten in Abhängigkeit von der verwendeten Batchgröße gemessen werden. Da beide verteilte Varianten lediglich am Batchende Nachrichten sendeten, war zu erwarten, dass sie deutlich besser

	238 Knoten	895 Knoten	1387 Knoten
Sensorknoten ID0			
batchweiser Ansatz	1176	1182	1416
Präfixbaum-Ansatz	16234	42362	64658
Sensorknoten ID1			
batchweiser Ansatz	2647	2687	2891
Präfixbaum-Ansatz	12383	19197	23555
Sensorknoten ID2			
batchweiser Ansatz	2279	2495	2731
Präfixbaum-Ansatz	6747	7299	7535

Tabelle 6.5: Speicherbedarf des FP^2-Tree auf den Sensorknoten (in Byte)

abschneiden als die Variante mit vollständig zentraler Verarbeitung. Aufgrund der Kettenstruktur sollte der Präfixbaum die wenigsten Nachrichten benötigen. Die Evaluierung erfolgte mittels der Simulationsumgebung, da in dieser ein einfaches Überwachen der versendeten Nachrichten möglich ist. Das Verfahren wurde mit 4 verschiedenen Batchgrößen evaluiert: 60, 100, 200 und 400 Datensätzen.

Die Abbildungen 6.18(a) und 6.18(b) zeigen die Gesamtanzahl der versendeten Nachrichten für die beiden drahtlos angebundenen Sensorknoten mit ID 1 und ID2. Die Nachrichtenanzahl umfasst dabei die von dem jeweiligen Knoten gesendeten Nachrichten (sowohl in Richtung Basisstation als auch über die Feedback-Komponente in Richtung Kettenende). Sehr gut zu erkennen ist die konstant hohe Anzahl an Nachrichten im Fall des reinen Sendens an die zentrale Instanz. Die beiden verteilten Verfahren benötigen jeweils deutlich weniger Nachrichten, wobei die Anzahl erwartungsgemäß mit steigender Batchgröße abnimmt. Ebenfalls sehr gut zu erkennen ist, dass der Knoten am Kettenende im Fall des Präfixbaum Ansatzes nahezu dieselbe Nachrichtenanzahl versendet wie im Fall des einfachen batchbasierten Ansatzes. Anders verhält sich dies auf dem zweiten Knoten in der Kette, welcher die Ergebnisse vom Kettenende erhält, diese in seinen eigenen lokalen FP^2-Tree einarbeitet und dann die Gesamtergebnisse weiterleitet. Hier zeigt sich der Präfixbaum-Ansatz insbesondere bei kleineren Batchgrößen als deutlich nachrichtenintensiver. Da der Knoten am Kettenende zwei Attribute erfasst, verwaltet dieser im Präfixbaum-Ansatz bereits einen einfachen FP^2-Tree. In Abhängigkeit der Verfeinerung der entsprechenden Intervalle kann dieser FP^2-Tree verhältnismäßig komplex sein, was im Weiteren eine entsprechend hohe Anzahl von zu versendenden Nachrichten zum Nachfolgeknoten nach sich zieht. Dementsprechend steigt auch die Anzahl der vom zweiten Knoten zum Knoten der Basisstation gesendeten Nachrichten. Erst bei einer Batchgröße von 400 Datensätzen wird der Präfixbaum-Ansatz effizienter als der einfache batchweise Ansatz. Aufgrund der großen Batchgröße kommt es vermutlich zu einer geringeren Verfeinerung, so dass weniger Knoten entstehen. Infolgedessen müssen weniger Nachrichten an die Folgeknoten in der Kette versendet werden.

Operator	Zeit in ms	elektrische Stromstärke in mA	konsumierte Energie in μJ
Einfügen eine Wertes			
batchweise Ansatz	11.9	6.6	259.2
Präfixbaum Ansatz	12.3	7.0	284.1
Batchende			
batchweise Ansatz	65.8	7.7	1672.0
Präfixbaum Ansatz	55.2	7.6	1384.4

Tabelle 6.6: Energieverbrauch für die beiden FP^2-Stream-In-Network-Operatoren

Im nächsten Test soll folgende These betrachtet werden:

These 17 *Die auf den Sensorknoten verfügbaren Speicherressourcen haben einen signifikanten Einfluss auf die teilweise Auslagerung der Verarbeitung des FP^2-Streams in das WSN.*

Hierfür wurden drei verschieden komplexe FP^2-Trees erzeugt. Die Anzahl der Knoten bewegte sich dabei zwischen 238 und 1387. Die Bestimmung des für die Verwaltung des FP^2-Trees benötigten Speichers pro Sensorknoten erfolgte auf Basis der Simulationsumgebung.

In Tabelle 6.5 ist der Speicherbedarf pro Knoten dargestellt. Im Fall des batchweisen Ansatzes steigt der Speicherbedarf mit der Komplexität der Datenstruktur nur langsam an. Sehr gut ist auch der Unterschied zwischen dem Knoten mit der ID0 und den anderen beiden Knoten zu erkennen. Da dieser lediglich ein Attribut überwacht, benötigt er auch nur halb so viel Speicher wie die anderen beiden Knoten. Die Tabelle zeigt auch, wie sich der Speicherbedarf im Fall des Präfixbaum-Ansatzes mit jedem zusätzlichen Knoten dramatisch erhöht. Hier erhöht sich der Speicherbedarf für eigene FP^2-Tree-Knoten um den Speicherbedarf für die FP^2-Tree-Knoten der Vorgänger in der Kettenstruktur. Auch die Größe des FP^2-Trees hat einen dramatischen Einfluss auf die Größe des benötigten Speichers. Im Fall des großen FP^2-Trees auf dem Sensorknoten mit der ID 0 überschreitet der benötigte Speicherbedarf bereits 64kB, was mehr ist, als viele Sensorknoten zu Verfügung stellen.

Die Tabelle zeigt weiterhin, dass – sofern vorab abgeschätzt werden kann, wie sich die Größe des FP^2-Tree entwickelt – auch eine Speicherabschätzung auf Basis von Erfahrungswerten möglich ist. Im Fall des batchbasierten Ansatzes entwickelt sich der benötigte Speicherplatz nahezu linear mit der Anzahl von Attributen und der Größe des FP^2-Trees.

Da die beiden verteilten Varianten batchbasiert sind, muss zwischen den Kosten für das Einfügen eines Wertes in den FP^2-Stream und den Kosten für die Auswertung am Batchende unterschieden werden. In Tabelle 6.6 sind die jeweiligen Energieverbräuche auf den Sensorknoten für einen mittelgroßen FP^2-Tree (895 Knoten) dargestellt. Statt der gemessenen $7344.8\mu J$ für das Senden jedes Datensatzes sind bei der In-Network-Verarbeitung des FP^2-Streams je nach Variante nur noch zwischen $259.2\mu J$

und $284.1\mu J$ notwendig. Erst am Batchende entstehen durch das Auslesen der Daten zusätzliche Kosten, die aber immer noch deutlich geringer ausfallen als die für das Versenden eines Datensatzes.

Die Experimente ergaben zwei wichtige Ergebnisse: (i) Der Ansatz der batchweisen Verarbeitung arbeitet in jedem Fall effizienter als die zentrale Variante und ist auch in Zusammenhang mit großen FP^2-Trees einsetzbar. (ii) Der Präfixbaum Ansatz ist zwar ebenfalls energieeffizienter als die zentrale Variante, seine Einsatzmöglichkeiten sind aber durch die Größe des FP^2-Trees beschränkt. Erst im Fall einer relativ kleinen Indexstruktur macht sich der erwartete Vorteil durch die geringere Nachrichtenanzahl bemerkbar. Die notwendige Einschwingphase führt bei diesem Ansatz außerdem dazu, dass er zu Beginn sehr kostenintensiv ist, so dass die vorhandenen Speicherressourcen auf den Sensorknoten unter Umständen aufgebraucht werden und es zu entsprechenden Verarbeitungsabbrüchen kommt. Hat sich die Anzahl der Knoten hingegen auf einem niedrigen Niveau eingependelt, so kann dieser Ansatz durchaus energieeffizienter sein als der primitive batchweise Ansatz. Für das Problem des Einschwingens sind mehrere Lösungen denkbar. Eine generelle Reduktion der Knotenanzahl im FP^2-Stream wäre eine Möglichkeit. Ein anderer Ansatz wäre eine hybride Lösung aus den beiden vorgestellten verteilten Ansätzen. Während der Einschwingphase wird zunächst mit dem batchweisen Ansatz begonnen. Nachdem sich die Knotenanzahl auf einem entsprechend niedrigen Niveau eingependelt hat, könnte anschließend auf den Präfixbaum-Ansatz gewechselt werden.

Fazit Im zurückliegenden Abschnitt wurde gezeigt, dass das im Rahmen dieser Arbeit entwickelte Verfahren zum quantitativen Frequent Pattern Mining für die Analyse von Datenströmen geeignet ist. Es wurde gezeigt, dass das eingeführte Maß der Informationsdichte für das Erzeugen interessanter Itemsets geeignet ist.

Einer der Nachteile des Verfahrens ist die unter Umständen große Anzahl an Knoten, welche in der Datenstruktur erzeugt werden. In den Beispielen übersteigt die Knotenanzahl während der initialen Phase teilweise sogar die Anzahl der im FP^2-Stream gespeicherten Transaktionen. Zwar pendelt sich die Knotenanzahl anschließend in allen Beispielen deutlich unterhalb der Transaktionszahl ein, dennoch sind teilweise deutlich über 100 Knoten für die Verwaltung der Daten nötig. Dies kann insbesondere im Umfeld der In-Network-Verarbeitung zu Problemen führen. Während der Beschreibung des Verfahrens in Abschnitt 5.3.2.4 wurde neben dem Entfernen von nicht mehr benötigten Knoten und dem Zusammenfügen von Knoten mit annähernd gleicher Informationsdichte die Reorganisation der Datenstruktur als mögliche Lösung vorgeschlagen. Weiterhin ist es denkbar, die bei der Rekombination gewonnenen Informationen zu den Itemgrenzen in den FP^2-Stream einfließen zu lassen. Zum Beispiel könnten Knoten, welche von den Itemgrenzen weiter entfernt liegen (sowohl innerhalb als auch außerhalb der Items), temporär verbunden werden, obwohl die geforderte Gleichheit der Informationsdichten nicht gegeben ist.

Ein weiterer Lösungsansatz zur Reduktion ist das Entfernen von Knoten, deren Häufigkeitswert unter einem definierten Schwellwert liegt. Hier könnte zum Beispiel der in [85]

vorgestellte Ansatz zum Pruning von Teilfenstern eingesetzt werden. Teile von in den Knoten eingebetteten Fenstern, welche nicht häufig sind und dies vermutlich auch nicht werden, können aus dem FP^2-Stream entfernt werden, was wiederum im besten Fall zum vollständigen Entfernen der Knoten führen kann. Bei diesem Ansatz besteht allerdings auch die Gefahr, Itemsets, welche möglicherweise häufig sind, nicht als solche zu erkennen. Im Zusammenhang mit dem FP^2-Stream spielt dies allerdings eine untergeordnete Rolle, da bei diesem Verfahren lediglich Daten mit einem beschränkten zeitlichen Horizont betrachtet werden.

6.3 Zusammenfassung

Im ersten Teil des zurückliegenden Kapitels wurden Details zur Implementierung von *AnduIN* beschrieben, welche über den grundlegenden Systementwurf aus Kapitel 3 hinausgehen. Die Beschreibung umfasste dabei neben der Nutzerschnittstelle auch zwei für die anschließende Evaluierung notwendige Aspekte. In Abschnitt 6.1.3.1 wurde ein Überblick über die verwendete Hardware-Plattform gegeben. Abschnitt 6.1.3.2 widmete sich der für die Implementierung und Evaluierung entwickelten Simulationsumgebung.

Im weiteren Verlauf des Kapitels wurden die in den vorangegangenen Kapiteln vorgestellten Verfahren und Techniken evaluiert. Mit Hilfe verschiedener Messungen bezüglich der beiden berücksichtigten Optimierungsziele Energie und Datendurchsatz wurde das in der Arbeit vorgestellte Kostenmodell analysiert. Weiterhin wurden die vorgestellten Operatoren hinsichtlich der geforderten Ziele und ihrer speziellen Eigenschaften evaluiert. Die Ergebnisse haben gezeigt, dass die gesetzten Zielen in fast allen Fällen erreicht werden konnten. Es wurde allerdings auch festgestellt, dass an einigen Stellen noch Potential für weitere Entwicklungen besteht.

Kapitel 7

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein neuartiger Ansatz zur teilweisen oder vollständigen Durchführung von Aufgaben in drahtlosen Sensornetzwerken beschrieben. Im nachstehenden Abschnitt 7.1 werden die wesentlichen Beiträge dieser Arbeit zusammenfassend dargestellt. Anschließend werden in Abschnitt 7.2 noch einmal offene Probleme zusammengetragen, welche sich im Kontext der Arbeit ergaben und interessante Fragestellungen für zukünftige Arbeiten bieten.

7.1 Zusammenfassung

Drahtlose Sensornetzwerke stellen eine Möglichkeit zur großflächigen Erfassung und Digitalisierung von Umgebungseigenschaften dar. Die Entwicklung entsprechender Hard- und Softwaresysteme in diesem Umfeld ist mit einer Vielzahl von Herausforderungen verbunden. Neben der Entwicklung geeigneter Hardware zählt insbesondere der Umgang mit den stark beschränkten Ressourcen zu den wesentlichen Herausforderungen. Die vorliegende Arbeit beschäftigte sich mit der effizienten Analyse von Sensordaten innerhalb und außerhalb des eigentlichen Sensornetzwerkes. Nach einem einleitenden Kapitel wurden existierende Lösungen aus den Bereichen der In-Network- und der Datenstromverarbeitung analysiert. Es zeigte sich dabei, dass der Forschungsschwerpunkt im Kontext von WSNs bisher im Bereich der Entwicklung von effizienten Algorithmen und Kommunikationslösungen lag. Für eine einfache deklarative Anfragebeschreibung steht zum gegenwärtigen Zeitpunkt mit TinyDB lediglich ein System zur Verfügung. Dessen monolitischer Aufbau erweist sich jedoch als problematisch bezüglich der Ausführung komplexer Analysen, wie sie zum Beispiel für Data Mining benötigt werden. Darüber hinaus zeigte sich, dass existierende DSMS nahezu beliebig komplexe Verfahren unterstützen können. Diese sind aber zumeist nicht für die In-Network-Verarbeitung konzipiert.

Diese Erkenntnisse motivierten die Entstehung von *AnduIN*, dem im Rahmen der vorliegenden Arbeit entwickelten System. *AnduIN* vereint dabei die Vorteile beider

Welten, INQP und DSMS, miteinander. Im Mittelpunkt des Entwurfes stand die systemübergreifende Verarbeitung von Sensordaten. Im Gegensatz zu existierenden Lösungen wurde auf den Sensorknoten eine dynamische Laufzeitumgebung eingesetzt. Diese wird vom System in Abhängigkeit der auszuführenden Aufgaben vollautomatisch konstruiert und auf die Sensorknoten verteilt. Einen ähnlichen Ansatz wie den in der Arbeit präsentierten verfolgt das 2009 in [68] vorgestellte System. Diese Arbeit zeigt, dass das hier bearbeitete Thema hochaktuell ist.

In *AnduIN* werden Aufgaben vom Nutzer mittels einer deklarativen Anfragesprache beschrieben und anschließend analysiert, optimiert und zur Ausführung gebracht. Hierzu wurde das aus dem Bereich der DBMS bekannte Modell der Anfrageverarbeitung um entsprechende Komponenten für die Datenstromverarbeitung und die Anfragedekomposition erweitert. Zum Zweck der plattformübergreifenden Anfrageevaluierung und -optimierung wurde das aus dem Bereich der Datenstromverarbeitung bekannte Operatorgraphenmodell auf die Sensornetzwerkkomponente des neu entwickelten Systems übertragen. Die Entscheidung darüber, welcher Anfrageplan letztendlich im System zur Ausführung kommt, wird in *AnduIN* auf Grundlage eines multidimensionalen Kostenmodells getroffen. Neben der Minimierung des Energieverbrauches im WSN wurde hierzu exemplarisch der Datendurchsatz auf der zentralen Komponente als zweites Optimierungsziel betrachtet.

Neben der Integration von Sensornetzwerken als mögliche Verarbeitungseinheiten wurden zusätzlich Erweiterungen für die Verarbeitung komplexer Verfahren geschaffen. Entsprechende Verfahren können dabei vollständig im WSN, vollständig auf der zentralen Komponente oder auf beiden Systemteilen bearbeitet werden. Es wurde gezeigt, dass sich die für die Nutzung dieser Verfahren notwendigen Erweiterungen in die Anfragesprache und das entwickelte Kostenmodell integrieren lassen.

Im weiteren Verlauf der Arbeit wurde die für *AnduIN* entwickelte Operatorbibliothek präsentiert. Neben einer Menge von Basisoperatoren wurde eine Vielzahl weiterer Operatoren hinzugefügt, welche insbesondere im Zusammenhang mit der Verarbeitung von Sensordaten von Interesse sind. So wurden grundlegende Funktionen für die Verarbeitung räumlicher Daten, wie sie zum Beispiel beim Lesen von einem GPS-Modul auftreten, geschaffen. Aus dem Bereich des Data Cleaning wurde neben der Integration eines einfachen Ausreißerverfahrens zusätzlich ein neuartiger Operator zur Erkennung von Bursts in trendbehafteten Daten entwickelt. Mit der Neuentwicklung eines Verfahrens zum Frequent Pattern Mining über quantitativen Attributen wurde zudem ein komplexes Verfahren präsentiert, welches die Möglichkeiten der teilweisen In-Network-Verarbeitung von *AnduIN* ausschöpft.

7.2 Ausblick

In der Arbeit wurden sowohl im Zusammenhang mit dem entwickelten System als auch mit den vorgestellten Operatoren bereits verschiedene offene Probleme identifiziert. Im Weiteren sollen grundlegende Herausforderungen zusammengetragen werden, welche

sich im Kontext der plattformübergreifenden Datenstromanalyse ergeben.

Anfragesprache Eine der grundlegenden Anforderungen an das in der Arbeit gestellte System war die Bereitstellung einer Möglichkeit zur deklarativen Beschreibung von Anfragen. Prinzipiell ist das der Anfragesprache zugrundeliegende Modell unabhängig von einer speziellen Anfragesprache. Aufgrund seiner Verbreitung bei DSMS wurde das mit STREAM eingeführte CQL als Sprachinterface integriert. Es hat sich aber gezeigt, dass CQL verschiedene Defizite aufweist. So mussten zum Beispiel für die Einbindung komplexer Operatoren und von WSNs als mögliche Datenquellen entsprechende Erweiterungen integriert werden.

Im Gegensatz zu DBMS, bei denen sich mit SQL ein Sprachstandard etablieren konnte, existiert für die datenstromverarbeitenden Systeme keine einheitliche Anfragesprache. Dies führt dazu, dass mit jedem neu entwickelten System häufig auch eine neue Anfragesprache bzw. Erweiterungen für eine existierende Anfragesprache präsentiert werden. Neben der Inkompatibilität der einzelnen Systeme bringt dies noch weitere Nachteile mit sich. Aufgrund der fehlenden einheitlichen Anfragesprache ist ein Vergleich verschiedener Lösungen untereinander schwierig. Es existieren zwar einfache Benchmarks wie zum Beispiel Linear Road [17] oder NEXMark [203]), welche den Vergleich verschiedener DSMS zum Ziel haben. Diese evaluieren aber oftmals nur einfache Teilaspekte dieser Systeme und sind auf die Verwendung einer gemeinsamen Anfragesprache wie CQL angewiesen. Dies schränkt sowohl den Rahmen der betrachteten Testszenarios als auch der Systeme stark ein. Gerade im Hinblick auf die Nutzung von DSMS im kommerziellen Umfeld sollten entsprechende einheitliche Standards geschaffen werden, welche zum einen eine einfache Migration auf andere Systeme ermöglichen und zum anderen deren Vergleichbarkeit hinsichtlich verschiedener Kriterien gewährleisten.

Skalierbarkeit Ein weiteres interessantes Problem ist die Skalierbarkeit des vorgestellten Ansatzes. Auf der Sensornetzwerkebene hängt die Skalierbarkeit nahezu vollständig von dem zugrundeliegenden Betriebssystem ab.

Anders verhält sich dies bei der zentralen Komponente. Hier finden alle Berechnungen auf einem einzigen System statt. Verwaltet man auf diesem eine Vielzahl von parallelen Anfragen, so führt dies möglicherweise zu Engpässen. Mit dem in Verbindung mit dem Kostenmodell vorgestellten Ansatz der Durchsatzoptimierung wurde bereits ein möglicher Ansatz zur Lösung dieses Problems präsentiert. Reichen die auf der zentralen Instanz vorhandenen Berechungsressourcen dennoch nicht für eine verlustfreie Datenanalyse aus, so werden alternative Ansätze benötigt. Interessant sind in diesem Zusammenhang die Möglichkeiten, die zum Beispiel moderne Grafikkarten bieten. Techniken wie CUDA (*Compute Unified Device Architecture*) oder OpenCL (*Open Computing Language*) stellen auch solchen Anwendungen, die nicht der graphischen Darstellung dienen, die Ressourcen der Grafikhardware zur Verfügung.

Grafikkarten bestehen häufig aus einer Vielzahl von Stream-Prozessoren, welche eine hochgradig parallele Ausführung von Berechnungen ermöglichen. Die Stream-

Prozessoren scheinen dabei insbesondere für die Verarbeitung von Datenströmen geeignet. So existieren bereits erste Ansätze, welche einfache Datenbankoperatoren auf Grafikkarten ausführen (zum Beispiel für PostgreSQL[1]). Mit der Verlagerung der Ausführung von Operatoren oder Anfrageteilen auf die Grafikkartenhardware ergeben sich allerdings auch neue Probleme. So ist beispielsweise die Kommunikation zwischen der Grafikkartenkomponente und dem restlichen DSMS mit einem erheblichen Aufwand verbunden. Das Problem ist dem in der vorliegenden Arbeit nicht unähnlich. Auch hier müssen vorab Entscheidungen über die Zweckmäßigkeit einer Auslagerung von Anfrageteilen in das WSN getroffen werden. Weiterhin müssen für die Bearbeitung von Operatoren auf Grafikkartenprozessoren zunächst existierende Algorithmen an die Architektur dieser angepasst werden.

Operatorkomplexität Mit der hier vorgestellten Arbeit wurde eine Möglichkeit zur Integration komplexer Operatoren in CQL-basierte Systeme geschaffen. Einen anderen interessanten Ansatz verfolgen die Autoren in [87]. Die von ihnen entwickelte UDF-ähnliche Anfragesprache WaveScript bietet die Möglichkeit, aus einfachen Algorithmen und Datenstrukturen komplexe Verfahren zu konstruieren. Basierend auf dieser Idee könnten zum Beispiel komplexe Data-Mining-Verfahren auf entsprechende primitive Verfahren zurückgeführt werden. In *AnduIN* wurde dieses Konzept bereits zum Teil umgesetzt. So wurde zum Beispiel das integrierte Clusterverfahren CLUEStream (siehe Tabelle 5.1) in die einfacheren Verfahren LBGU und MicroCluster zerlegt, welche für die Ausführung des CLUEStream-Verfahrens geeignet kombiniert wurden. Auf ähnliche Weise wurde mit dem vorgestellten Operator zur adaptiven Burst-Erkennung verfahren. Dieser ließ sich in die Teile exponentielle Glättung und einfache Burst-Erkennung zerlegen, so dass mit diesen prinzipiell drei Operatoren zur Verfügung stehen. Die Zerlegung in Primitive bietet neben der Möglichkeit zur Mehrfachverwendung und Neukombination gerade im Zusammenhang mit der Operatorauslagerung Potential. So kann eine Teilung von komplexen Operatoren für die partielle In-Network-Verarbeitung eventuell entlang der primitiven Operatoren vorgenommen werden.

Semantic Sensor Web Zur Unterstützung des einfachen Anlegens von Datenströmen und zur Visualisierung von Ergebnisdatenströmen wurde in der vorliegenden Arbeit eine entsprechende GUI entwickelt. Für die Verwaltung kleiner einfacher Aufgaben ist diese gut geeignet. Problematisch gestaltet sich hingegen die Verwaltung, Verknüpfung und Auswertung vieler Tausende oder Millionen gleichzeitiger Datenströme. Gerade in den letzten Jahren hat sich die durch Sensoren erzeugte Datenmenge explosionsartig vervielfacht. Es wird geschätzt, dass allein das durch Sensoren in Fahrzeugen im Jahr 2015 erzeugte Datenaufkommen 6.4TBps erreicht[2]. Zum Vergleich: Im Jahr 2010 hatte der weltweit verkehrsreichste Internetdatenknoten DE-CIX in Frankfurt einen

[1] http://highscalability.com/scaling-postgresql-using-cuda

[2] M. Raskino, J. Fenn, and A. Linden: Extracting Value From the Massively Connected World of 2015. Gartner Research, April 2005.

Spitzendurchsatz von 1.5TBps[3].

Eine der größten Herausforderungen der nächsten Jahre wird es sein, diese von Sensoren auf der ganzen Welt erzeugten Daten auszuwerten und sinnvoll miteinander und mit Inhalten anderer Quellen zu verknüpfen. Plattformen wie zum Beispiel *Sensor Web*[4] oder GSN[5] stellen entsprechende Infrastrukturen bereit, welche dies ermöglichen sollen. Zu den wesentlichen Aufgaben im Kontext solcher Systeme zählt dabei die Vielzahl von möglichen Datenquellen. Um die Interoperabilität zwischen verschiedenen Datenquellen zu gewährleisten, wird mittlerweile immer stärker auf Techniken aus dem Web 2.0 (beispielsweise auf Tagging) zurückgegriffen. Sensordaten werden automatisch oder manuell mit zusätzlichen Informationen (zum Beispiel der geografischen Lage der erzeugenden Sensoren) annotiert.

[3]http://www.de-cix.net/content/network.html
[4]http://www.sensorweb-alliance.org/
[5]http://sourceforge.net/apps/trac/gsn/

Anhang A

Exponentielle Glättung

Die exponentielle Glättung (engl. *exponential smoothing*) ist ein statistisches Vorhersageverfahren, welches auf Basis vergangener Werte kurzfristig zukünftige Werte vorhersagen kann, unabhängig von der Datenverteilung der gemessenen Werte. Bei der exponentiellen Glättung werden im wesentlichen drei Verfahren unterschieden [34], welche im Folgenden kurz vorgestellt werden sollen.

A.1 Einfache exponentielle Glättung

Die einfache exponentielle Glättung findet Anwendung im Bereich der Kurzzeit-Vorhersagen und bei statischen Zeitreihen. Sie ist wie folgt definiert:

$$F_t = \alpha \cdot y_t + (1 - \alpha) \cdot F_{t-1}$$

S_t beschreibt den Vorhersagewert, wobei α, $0 < \alpha < 1$ ein Gewicht ist, welches den Einfluss neuerer Werte wiedergeben soll.

Löst man die Rekursion durch wiederholtes Einsetzen der Gleichung S_t auf, so erhält man

$$\begin{aligned} F_t &= \alpha \cdot (y_t + (1-\alpha) \cdot y_{t-1} + (1-\alpha)^2 \cdot y_{t-2} \\ &\quad + (1-\alpha)^3 \cdot y_{t-3} + \cdots) + (1-\alpha)^t \cdot y_0 \end{aligned}$$

A.2 Holt

Eine Erweiterung der einfachen exponentielle Glättung ist die Holt-Glättung oder auch doppelte exponentielle Glättung. Die Holt-Glättung berücksichtigt auch Trends und ist wie folgt definiert:

$$\begin{aligned} F_t &= \alpha \cdot y_t + (1-\alpha)(F_{t-1} + b_{t-1}) \\ b_t &= \gamma \cdot (F_t - S_{t-1}) + (1-\gamma)(b_{t-1}) \end{aligned}$$

α und γ, mit $0 < \alpha, \gamma < 1$ sind wiederum Gewichte, welche den Einfluss neuerer Werte bzw. des Trends angeben. Ein mögliches Verfahren zur Bestimmung dieser ist das Marquardt-Verfahren [154]. Zusätzlich zur Vorhersage wird hier noch der Trend b_t der Zeitreihe ermittelt, welcher bei der Vorhersage des nächsten Zeitpunktes berücksichtigt wird.

A.3 Holt-Winter

Obwohl es nicht in dieser Arbeit verwendet wurde, soll hier der Vollständigkeit halber noch die dritte exponentielle Glättung bzw. das Holt-Winters-Verfahren erwähnt werden. Bei diesem werden zusätzlich zu den Trends auch noch Perioden, d.h. zeitliche Wiederholungen, wie sie zum Beispiel bei Sommer-Winter Schwankungen auftreten, berücksichtigt.

$$\begin{aligned} F_t &= \alpha \cdot \frac{y_t}{I_{t-L}} + (1-\alpha)(F_{t-1} + b_{t-1}) \\ b_t &= \gamma \cdot (S_t - S_{t-1}) + (1-\gamma) \cdot b_{t-1} \\ I_t &= \beta \cdot \frac{y_t}{S_t} + (1-\beta) \cdot I_{t-L} \end{aligned}$$

α, β und γ, mit $0 < \alpha, \beta, \gamma < 1$ sind wiederum Gewichte, welche den Einfluss neuer Werte, den Trend und die Stärke der Periode bestimmen. S_t ist der Vorhersagewert, b_t bezeichnet die Trendabschätzung, I_t ist der saisonale Index, und L ist die Länge der Periode. Eines der größten Probleme dieses Verfahrens ist die Identifizierung der Länge der Periode, welche zu Beginn der Abschätzung bekannt sein muss.

Die vorgestellten Vorhersageverfahren können auch als ein Speziallfall des in der Statistik weit verbreiteten ARIMA-Modells [162] angesehen werden. Die einfache exponentielle Glättung entspricht dem ARIMA(0,1,1)-Modell; die doppelte exponentielle Glättung dem ARIMA(0,2,2)-Modell; und das Holt-Winter Verfahren dem ARIMA(0,3,3)-Modell. Einer der wesentlichen Vorteile des ARIMA-Modells ist die automatische Berechnung der Gewichte während der Schätzung des gleitenden Durchschnitts des ARIMA-Modells.

Anhang B

Grafische Nutzeroberfläche

Die Bilder in den Abbildungen 2.1, Abbildungen 2.2 und Abbildungen 2.3 zeigen verschiedene Screenshots der im Zusammenhang mit *AnduIN* entwickelten grafischen Nutzeroberfläche.

Abbildung 2.1 zeigt exemplarisch das Anlegen einer virtuellen Netzwerkquelle. Die Bibliothek auf der linken Seite enthält dabei u.a. eine Liste der bei *AnduIN* registrierten Operatoren, welche sich auf der Arbeitsfläche zu einer logischen Netzwerktopologie verknüpfen lassen. Beim Absenden wird die Liste automatisch in die in Abschnitt 3.3.1.1 erläuterte flache Darstellung überführt und bei *AnduIN* registriert.

Die Abbildungen 2.2 und Abbildungen 2.3 zeigen die Möglichkeit der Visualisierung des Ergebnisdatenstromes. In Abbildung 2.2 wird ein neues Diagramm über den Ergebnisdaten eines Datenstromes definiert. In Abbildung 2.3 ist zusätzlich eine geografische Karte mit den aktuell registrierten Sensorknoten zu sehen. Beim Anklicken eines oder mehrerer diese Sensoren erfolgt eine zusätliche Filterung über den Datenstromergebnissen. Soweit eine eindeutige Zuordnung möglich ist, werden anschließend ausschließlich

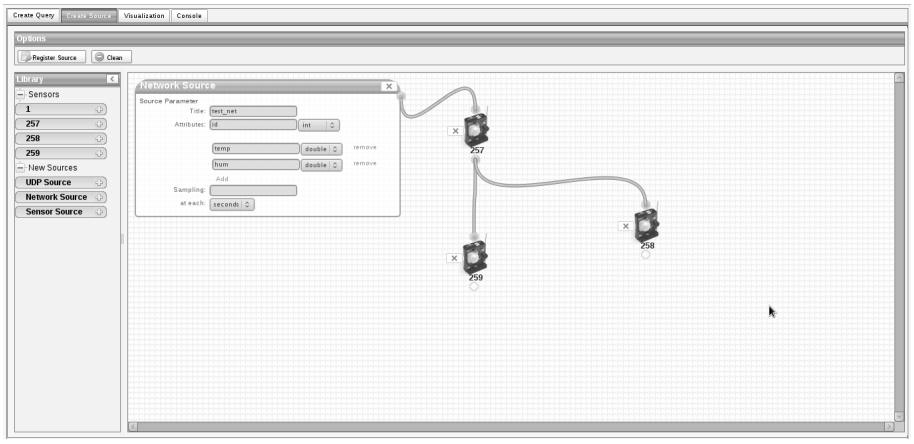


Abbildung 2.1: Netzwerkdatenquelle anlegen

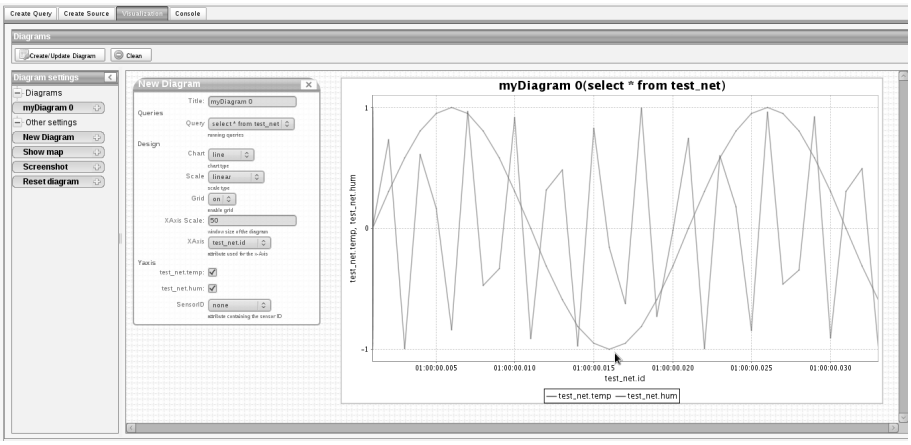


Abbildung 2.2: Datenstromvisualisierung

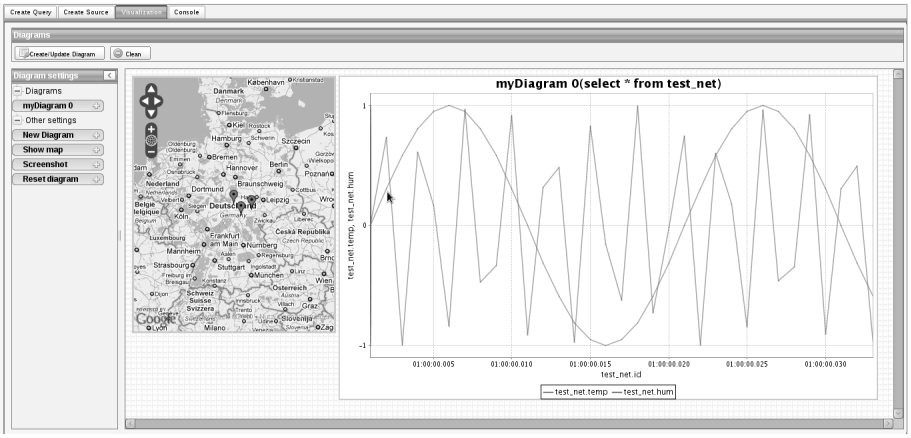


Abbildung 2.3: Datenstromvisualisierung inklusive einem räumlichen Filter unter Verwendung von Google-Maps

die Ergebnisse der ausgewählten Datenquellen visualisiert.

Anhang C

AnduIN Befehlsreferenz

Der folgende Abschnitt gibt einen Überblick über die wesentlichen Kommandozeilen-Befehle zur Bedienung von *AnduIN* und eine Beschreibung der verwendeten Anfragesprache. Jeder Befehl wird hierbei zunächst abstrakt eingeführt und anschließend an einem konkreten Beispiel erklärt.

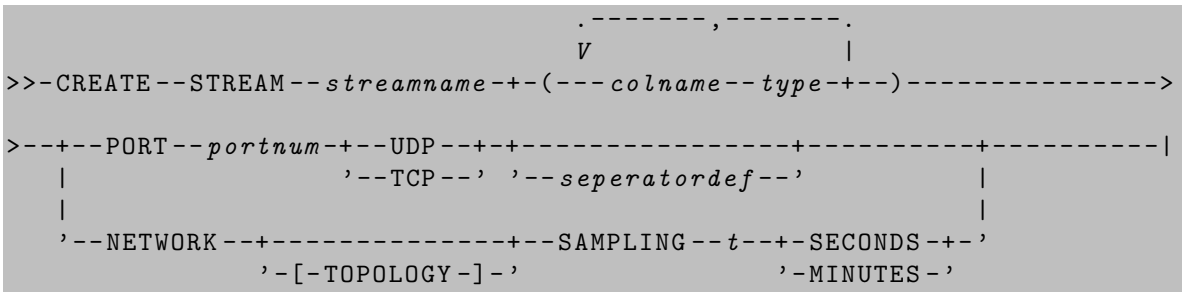
- `CREATE` bezeichnet ein Terminal
- *`colname`* einen Bezeichner
- */`select_clause`/* einen Abschnitt, welcher im Weiteren separat beschrieben wird

C.1 Create

Registrieren von Datenquellen.

C.1.1 Datenstrom registrieren

Einen neuen Datenstrom registrieren. Datenströme können von einem Netzwerkport gelesen werden, logische Sensornetzwerke sein oder logische Ströme über bereits existierenden Datenströmen.



Beispiele:

```
CREATE STREAM mystream (x int) PORT 8000 UDP;
CREATE STREAM mynetwork (id int, temp double)
    NETWORK SAMPLING 10 SECONDS;
CREATE STREAM mynetwork (id int, temp double)
    NETWORK SAMPLING 10 SECONDS;
```

C.1.2 Sicht über Datenstrom anlegen

Erzeugt einen logischen Datenstrom über bereits registrierten Datenströmen.

```
>>-CREATE--STREAM--streamname--AS--|select_statement|----------------|
```

Beispiel:

```
CREATE STREAM viewname AS SELECT * from mystream where value > 10;
```

C.1.3 Tabelle anlegen

Legt eine statische Tabelle in einem relationalen Backend an.

```
                                  .-------,-------.
                                  V               |
>>-CREATE--TABLE--tablename-+-(---colname--type-+--)-----------------|
```

Beispiel:

```
CREATE TABLE table (x int, y string);
```

C.1.4 Listener anlegen

Erzeugt einen Listener für die Client-Server-Kommunikation.

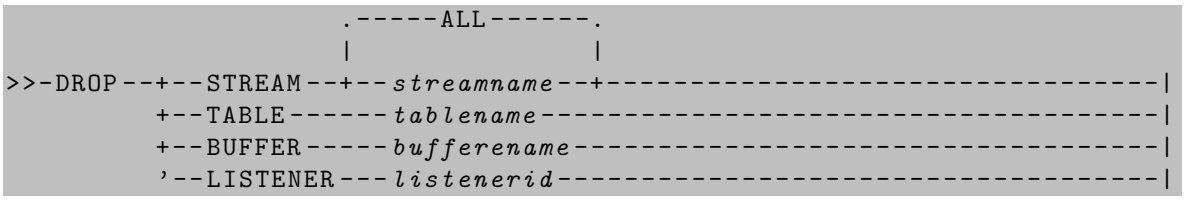
```
>>-CREATE--LISTENER--WITH--NETWORK--CONNECTION--USING--+--UDP--+------>
                                                      '--TCP--'
>--VERSION--+--4--+--PROTOCOL--ipadresse--:--portnumber--------------|
            '--6--'
```

Beispiel:

```
CREATE LISTENER WITH NETWORK CONNECTION USING UDP
       VERSION 4 PROTOCOL 141.24.33.18:10000
```

C.2 Drop

Löscht registrierte Datenquellen. Die zu löschende Datenquelle darf aktuell nicht durch eine laufende Anfrage blockiert sein.

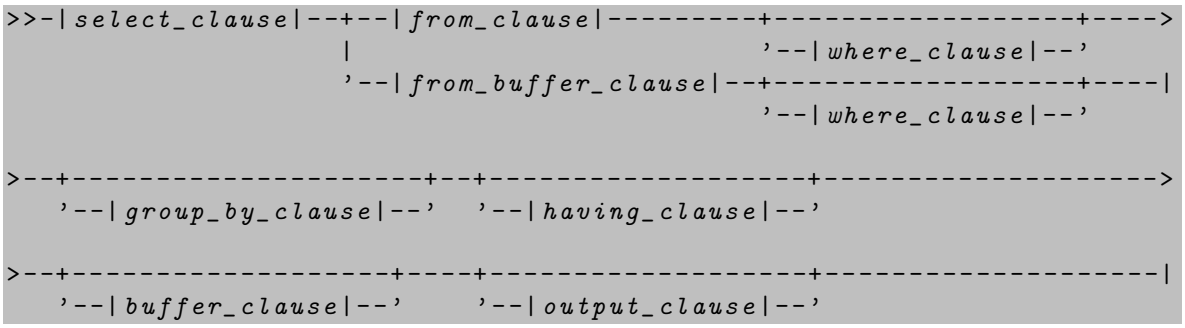


Beispiel:

```
DROP STREAM mystream;
DROP STREAM mynetwork;
DROP STREAM ALL;
DROP TABLE mytable;
```

C.3 Select

Anfrage über einer oder mehreren Datenquellen.

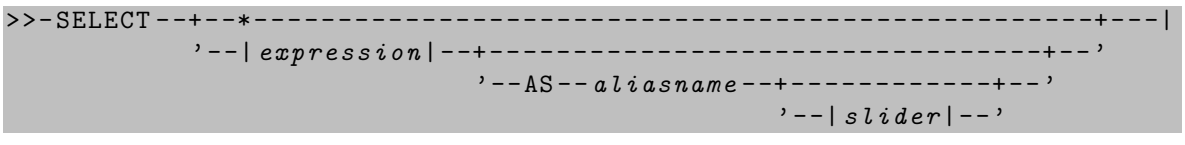


Beispiel:

```
SELECT * FROM mystream;
SELECT avg(temp) slide 30 minutes FROM mynetwork [RANGE 1 HOURS];
```

C.3.1 select-clause

Definiert, wie ein Ausgabetupel aufgebaut ist.



C.3.2 expression

Ein Expression beschreibt einen Wert, welcher verschiedene Formen annehmen kann.

Es stehen folgende Operatoren *operator* zur Verfügung:

- arithmetische Operatoren: $+, -, *, /, \%$
- logische Operatoren *and* und *or*

Weiterhin stehen über folgende Datentypen nachstehende Funktionen zur Verfügung:

- Integer:
  - *year, month, day* konvertiert einen Integer in das jeweilige Datum
  - *systime* liefert die Systemzeit
- Double:
  - *floor, ceil, random, power, abs*
- Strings:
  - *contains* testet, ob ein String einen definierten String enthält
  - *hashtags* entfernt alle nicht Hashtags aus dem Datenstrom
  - *to_lower, to_upper* konvertiert einen String in Klein- bzw. Großbuchstaben
- Geodaten:
  - *geo_loc(lat,lon,'func', rad)* Reverse Geocodeing [1] – liefert eine Platz im Umkreis der spezifierten Position

C.3.3 slider

Die Ausgabe eines Tupels soll im Abstand von *time* Zeiteinheiten erfolgen.

[1] greift auf http://www.geonames.org/ zu

C.3.4 from-clause

Beschreibt den Datenstrom, über dessen Attributen die Analyse durchgeführt wird.

C.3.5 from-buffer-clause

Fragt die Tupel des definierten Puffers ab.

C.3.6 where-clause

Definiert, welche Tupel der Ergebnissmenge angehören und welche nicht.

C.3.7 group-by-clause

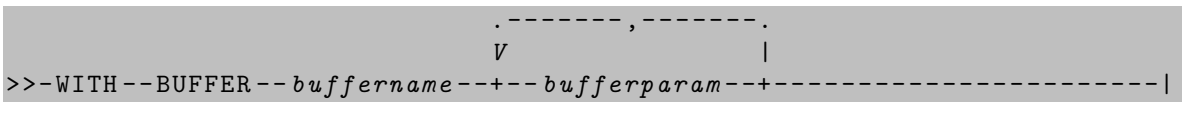
Gruppiert Tupel über der definierten Spalte.

C.3.8 having-clause

Die Ergebnismenge einer Gruppierung muss der definierten Having-Clause genügen, sonst ist sie nicht Teil der Ausgabemenge.

C.3.9 buffer-clause

Definiert einen Ringpuffer zum Zwischenspeichern von Ergebnissen der Anfrage. Die Ergebnisse können anschließend über eine Pufferanfrage abgefragt werden. Der Puffer wird so lange mit Daten gefüllt, bis die Anfrage gestoppt wird.

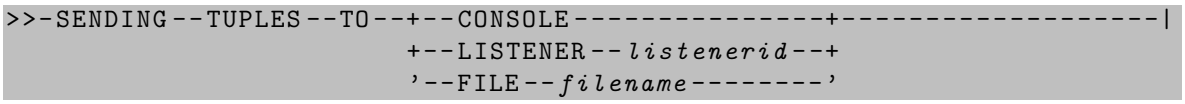


Möglicher Pufferparameter *bufferparam*:

- BUFSIZE Anzahl an Tupeln die gepuffert werden sollen

C.3.10 output-clause

Definiert das Ausgabeziel für den Ergebnisstrom einer Anfrage.

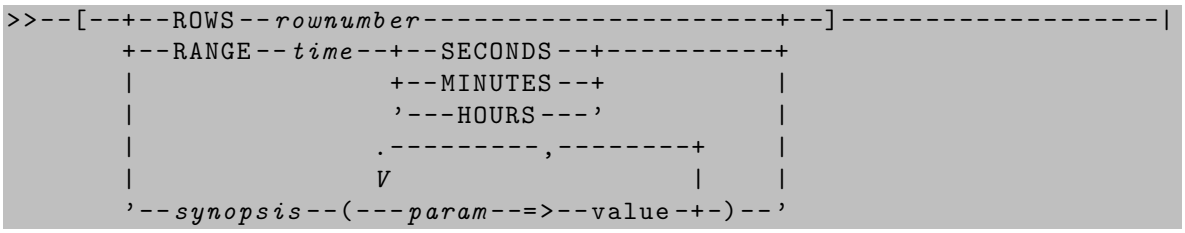


Mögliche Ausgabeziele:

- CONSOLE Die Ausgabe erfolgt auf der Konsole (Standard).
- LISTENER Die Ausgabe wird an den definierten Listener weitergeleitet.
- FILE Der Ausgabestrom wird in die definierte Datei umgeleitet.

C.3.11 synopsis-clause

Algebraischer Operator zur Integration komplexer Operatoren.



- *synopsis* Synopsenname
- *param* Parametername, welcher den Synopsen-Operator konfiguriert
- *value* Wert für den definierten Parameter

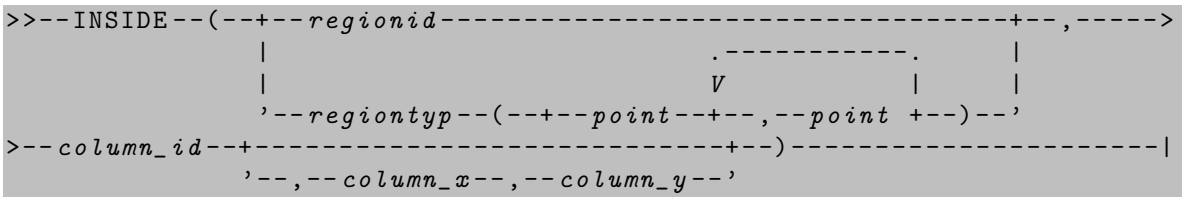
Mögliche Synopsen sind:

- rules - Erzeugt aus dem gegebenen Datenstrom Assoziation Rules
- burst - Adaptive Burst Erkennung
- fppstream - Quantitatives Frequent Pattern Mining
- outlier - Ausreiser-Erkennung
- smooth - Vorhersage (Exponentielle Gättung 1.,2. und 3. Ordnung)
- cluster - CLUStream-Clustering

C.3.12 spatial-clause

Definition von Filtern über Geo-Objekten.

Es werden die in Abschnitt 5.1 beschriebenen räumlichen Operatoren (inside, within-distance und nearest neighbor) unterstützt.

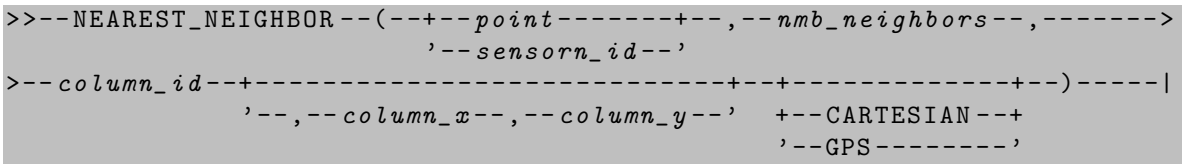


Es werden nachstehende Geo-Objekte unterstützt: *regiontyp* beschreibt den verwendeten Regionentyp. Zur Auswahl stehen

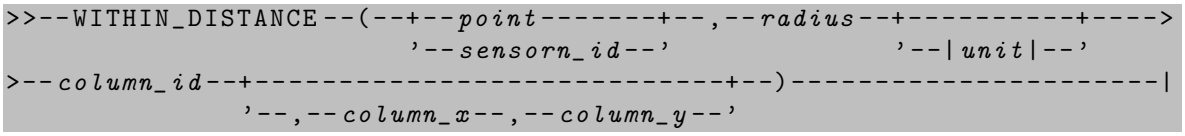
- *sdt_rect* Rechteck, definiert über die Punkte links oben und rechts unten
- *sdt_polygon* Polygon, definiert über die Angabe aller Eckpunkte
- *sdt_line* Linienzug, definiert über die Angabe aller Punkte des Linienzuges

Der Geo-Operator hat die folgenden Parameter:

- *regionid* die ID einer zuvor registrierten Region
- *point* ein (Eck-)Punkt, welcher das räumliche Objekt beschreibt
- *column_id* die ID des Attributes im Datenstrom, welche für einen registierten Sensorknoten dessen ID enthält
- *column_x, column_y* die IDs der Attribute, welche die x-bzw. die y-Koordinaten des beweglichen Objektes beinhalten



- *point* die Koordinaten des Punktes, welcher als Pivot-Element verwendet werden soll
- *sensorn_id* die ID des Knotens, welcher als Pivot-Element verwendet werden soll
- *nmb_neighbors* Anzahl der Objekte, welche die Nachbarschaft beinhalten soll



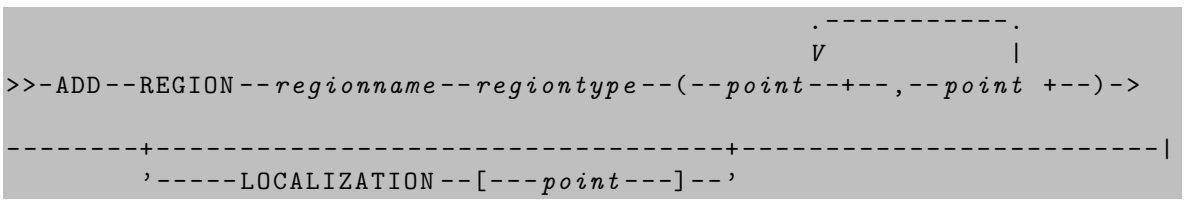
- *radius* bezeichnet den Radius um das Pivot-Element, in welchem sich ein Objekt befinden muss, um den Filter zu passieren
- *unit* bezeichnet die Einheit, in welcher der Radius definiert wird (mögliche Werte: *m*-Meter und *km*-Kilometer, voreingestellt *m*). Die Verwendung einer Maßeinheit führt automatisch zur Verwendung des sphärischen Koordinatensystem.

C.4 Geo-Objekte

Geo-Objekt zur Integration in Anfragen mit räumlichem Bezug verwalten.

C.4.1 Registrieren

Geo-Objekt Anlegen



Es werden nachstehende Geo-Objekte unterstützt: *regiontyp* beschreibt den verwendeten Regionentyp. Zur Auswahl stehen:

- *sdt_rect* Rechteck, definiert über die Punkte links oben und rechts unten
- *sdt_polygon* Polygon, definiert über die Angabe aller Eckpunkte
- *sdt_line* Linienzug, definiert über die Angabe aller Punkte des Linienzuges

Der Geo-Operator hat die folgenden Parameter:

- *regionname* beschreibt den Objektnamen, unter dem das Objekt in Anfragen integriert werden kann
- *point* ein (Eck-)Punkt, welcher das räumliche Objekt beschreibt

C.4.2 Löschen

Entfernt das registrierte Geo-Objekt *regionname* aus *AnduIN*.

```
>>-DELETE--REGION--regionname-----------------------------------------|
```

C.5 Statische Sensorknoten registrieren

Sensorknoten mit Schemata und Geolokation bei *AnduIN* registrieren.

```
                                .-------,-------.
                                V               |
>>-ADD--SENSOR--sensorname--(---colname--type-+-)--------------------->

--------+-----------------------------------+-------------------------|
        '-----LOCALIZATION--[---point---]--'
```

Sensorknoten aus Metadaten-Katalog entfernen.

```
>>-DELETE--SENSOR--sensorname------------------------------------------|
```

Beispiele:

```
ADD SENSOR id1 (temp double, hum double) LOCALIZATION [(5.6,7.8)];
DELETE SENSOR id1;
```

Das erste Kommando registriert einen Sensorknoten mit ID *id1* und den Attributen *temp* und *hum* sowie den GPS Koordinaten (default) $(5.6, 7.8)$. Das delete Kommando entfernt den Sensor wieder aus dem Metadaten-Katalog (solange dieser nicht Teil einer laufenden Anfrage ist).

C.6 Stop

Stoppt die Ausführung laufender Anfragen bzw. das Senden an registrierte Listener.

Beispiele:

```
STOP 1;
STOP SENDING TUPLES TO LISTENER 1 FROM QUERY 2;
```

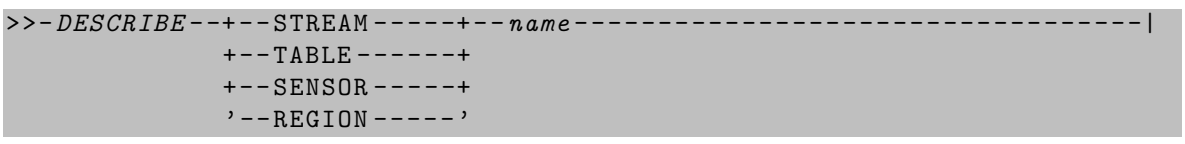
Das erste Kommando stoppt die Verarbeitung der Anfrage mit ID 1. Das zweite Kommando stoppt das Versenden der Ergebnisse der Anfrage mit ID 2 an den Listener mit ID 1.

C.7 Show

Gibt eine Liste aller registrierten Datenquellen, registrierten Sensoren, registrierten Regionen bzw. registrierten Listenern.

C.8 Describe

Gibt detailierte Informationen zu einer registrierten Datenquelle, einem registrierten Sensor oder einer registrierten Region.



Literaturverzeichnis

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. h. Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and St. Zdonik. The design of the borealis stream processing engine. In *CIDR*, pages 277–289, Asilomar, CA, 2005.

[2] D. J. Abadi, W. Lindner, S. Madden, and J. Schuler. An integration framework for sensor networks and data stream management systems. In *VLDB*, 2004.

[3] D. J. Abadi, S. Madden, and W. Lindner. Reed: Robust, efficient filtering and event detection in sensor networks. In *VLDB*, pages 769–780, 2005.

[4] G. D. Abowd. Ubiquitous computing: Research themes and open issues from an applications perspective. Technical report, 1997.

[5] M. I. Afzal, Sh. Seoyong, W. Mahmood, and S. M. Sajid. Optical wireless recharging model for wireless sensor nodes by using corner cube retroreflectors (ccrs). In *ICHIT '09*, pages 626–631, New York, NY, USA, 2009. ACM.

[6] Ch. Aggarwal. *Data Streams: Models and Algorithms.* Advances in Database Systems. Springer; 1. edition, 2007.

[7] Ch. C. Aggarwal, J. Han, J. Wang, and Ph. S. Yu. A Framework for Clustering Evolving Data Streams. In *VLDB*, pages 81–92, 2003.

[8] Ch. C. Aggarwal, J. Han, J. Wang, and Ph. S. Yu. A Framework for Projected Clustering of High Dimensional Data Streams. In *VLDB*, pages 852–863, 2004.

[9] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD 1993*, pages 207–216, 1993.

[10] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *VLDB 1994*, pages 487–499, 1994.

[11] Y. Ahmad, A. Jhingran, B. Berg, A. Maskey, W. Xing, O. Papaemmanouil, Y. Xing, M. Humphrey, A. Rasin, and St. Zdonik. Distributed operation in the borealis stream processing engine. In *SIGMOD*, 2005.

[12] L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King, Y. Park, and Ch. Venkatramani. Spc: A distributed, scalable platform for data mining. In *DM-SSP*, 2006.

[13] A. Arasu, B. Babcock, Sh. Babu, J. Cieslewicz, K. Ito, R. Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. Springer, 2004.

[14] A. Arasu, B. Babcock, Sh. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. Stream: the stanford stream data manager (demonstration description). In *SIGMOD*, pages 665–665, New York, NY, USA, 2003. ACM.

[15] A. Arasu, Sh. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical report, University of Stanford, 2003.

[16] A. Arasu, Sh. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

[17] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear road: a stream data management benchmark. In *VLDB*, pages 480–491. VLDB Endowment, 2004.

[18] Y. Aumann and Y. Lindell. A Statistical Theory for Quantitative Association Rules. *Journal of Intelligent Information Systems*, 20(3):255–283, 2003.

[19] R. Avnur and J. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *SIGMOD*, pages 261–272, 2000.

[20] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In *SIGMOD*, pages 253–264, 2003.

[21] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[22] B. Babcock, Sh. Babu, M. Datar, R. Motwani, and D. Thomas. Operator scheduling in data stream systems. *The VLDB Journal*, 13(4):333–353, 2004.

[23] B. Babcock, M. Datar, and R. Motwani. Load shedding techniques for data stream systems. In *Mpds 2003*, San Diego, California, USA, 2003.

[24] Sh. Babu and P. Bizarro. Adaptive query processing in the looking glass. In *CIDR 2005*, January 2005.

[25] Sh. Babu, R. Motwani, K.h Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. Technical Report 2003-69, Stanford InfoLab, 2003.

[26] S. D. Bay and M. Schwabacher. Mining Distance-Based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule. In *KDD 2003, Washington, D.C., USA*, pages 29–38, 2003.

[27] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In H. Garcia-Molina and H. V. Jagadish, editors, *SIGMOD*, pages 322–331, 1990.

[28] F. Beier. Entwicklung und Implementierung einer Kommunikationsschnittstelle und eines Pufferoperators fur das Stromdatenverarbeitungssystem AnduIN, 2009. Studienarbeit.

[29] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[30] J. Beringer. *Online-Data-Mining auf Datenströmen: Methoden zur Clusteranalyse und Klassifikation*. 2007.

[31] J. Beringer and E. Hüllermeier. Online clustering of data streams. Technical report, 2003.

[32] L. Berti-Équille. Contributions to Quality-Aware Online Query Processing. *IEEE Data Engineering Bulletin*, 29(2):32–42, 2006.

[33] Sh. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, Ch. Gruenwald, A. Torgerson, and R. Han. Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Network Applications*, 10(4):563–579, 2005.

[34] B. Billah, M. L. King, R. D. Snyder, and A. B. Koehler. Exponential smoothing model selection for forecasting. Monash econometrics and business statistics working papers, Monash University, Department of Econometrics and Business Statistics, 2005.

[35] P. Bizarro, Sh. Babu, D. DeWitt, and J. Widom. Content-based routing: different plans for different data. In *VLDB*, pages 757–768. VLDB Endowment, 2005.

[36] A. Bolles, M. Grawunder, J. Jacobi, D. Nicklas, and H.-J. Appelrath. Odysseus: Ein framework für maßgeschneiderte datenstrommanagementsysteme. In *GI Jahrestagung*, pages 2000–2014, 2009.

[37] Ch. Borgelt. An implementation of the fp-growth algorithm. In *OSDM*, pages 1–5, New York, NY, USA, 2005. ACM.

[38] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Icde'01*, pages 421–432, Washington, DC, USA, 2001. IEEE Computer Society.

[39] I. Botan, Y. Cho, R. Derakhshan, N. Dindar, A. Gupta, L. M. Haas, K. Kim, Ch. Lee, G. Mundada, M.-Ch. Shan, N. Tatbul, Y. Yan, B. Yun, and J. Zhang. A demonstration of the maxstream federated stream processing system. In *ICDE*, pages 1093–1096, 2010.

[40] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. LOF: Identifying Density-Based Local Outliers. In *SIGMOD*, pages 93–104, 2000.

[41] Ph. Buonadonna, D. Gay, J. M. Hellerstein, W. Hong, and S. Madden. Task: Sensor network in a box. In *EWSN*, 2005.

[42] M. Cammert, Ch. Heinz, J. Krämer, M. Schneider, and B. Seeger. A status report on xxl - a software infrastructure for efficient query processing. *Data Engineering Bulletin*, 26:2003, 2003.

[43] M. Cammert, J. Kramer, B. Seeger, and S. Vaupel. A cost-based approach to adaptive resource management in data stream systems. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):230–245, 2008.

[44] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, J. Thomas, J. H, and E. L. Wimmers. Towards heterogeneous multimedia information systems: The garlic approach. In *In RIDE-DOM*, pages 124–131, 1995.

[45] D. Carney, U. Çetintemel, A. Rasin, S. Zdonik, M. Cherniack, and M. Stonebraker. Operator scheduling in a data stream manager. In *VLDB*, 2003.

[46] D. Carney, U. Centintemel, A. Rasin, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Monitoring streams: A new class of data management applications. In *VLDB*, pages 215–226. VLDB Endowment, 2002.

[47] S. Chandrasekaran, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, 2003.

[48] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: continuous dataflow processing. In *SIGMOD*, pages 668–668, New York, NY, USA, 2003. ACM.

[49] S. Chaudhuri, R. Motwani, and V. R. Narasayya. Random Sampling for Histogram Construction: How much is enough? In *SIGMOD*, pages 436–447, 1998.

[50] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. pages 379–390, 2000.

[51] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. pages 323–334, 2002.

[52] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and St. Zdonik. Scalable distributed stream processing. In *CIDR*, 2003.

[53] E. Chervakova, D. Klan, and T. Rossbach. Energy-optimized sensor data processing. In *EUROSSC*, pages 35–38, 2009.

[54] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 26(1):64–69, 1983.

[55] O. Cooper, A. Edakkunni, M. Franklin, W. Hong, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, and E. Wu. HiFi: A unified architecture for high fan-in systems. In *VLDB*, pages 1357–1360. Demo, 2004.

[56] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco. Programming wireless sensor networks with the teenylime middleware. In *Middleware*, pages 429–449, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[57] Ch. Cranor, Th. Johnson, and O. Spataschek. Gigascope: a stream database for network applications. In *SIGMOD*, pages 647–651, 2003.

[58] D. E. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo. A network-centric approach to embedded software for tiny devices. *Lecture Notes in Computer Science*, 2211:114–130, 2001.

[59] J. V. d. Bercken, B. Blohsfeld, J.-P. Dittrich, J. Krämer, T. Schäfer, M. Schneider, and B. Seeger. XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries. In *VLDB*, pages 39–48, 2001.

[60] B. Dageville and M. Zait. Sql memory management in oracle9i. In *VLDB*, pages 962–973. VLDB Endowment, 2002.

[61] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows (extended abstract). In *SODA*, pages 635–644, 2002.

[62] M. Demirbas. Wireless sensor networks for monitoring of large public buildings. *Computer Networks*, 46:605–634, 2005.

[63] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *P2P*, page 32, Washington, DC, USA, 2003. IEEE Computer Society.

[64] A. Deshpande. An initial study of overheads of eddies. *SIGMOD Rec.*, 33(1):44–49, 2004.

[65] A. Deshpande, J. M. Hellerstein, and V. Raman. Adaptive query processing: why, how, when, what next. In *SIGMOD*, pages 806–807, New York, NY, USA, 2006. ACM.

[66] A. Deshpande, Z. G. Ives, and V. Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.

[67] F. Dressler. A study of self-organization mechanisms in ad hoc and sensor networks. *Comput. Commun.*, 31(13):3018–3029, 2008.

[68] F. Dressler, R. Kapitza, M. Daum, M. Strübe, W. Schröder-Preikschat, R. German, and Kl. Meyer-Wegener. Query processing and system-level support for runtime-adaptive sensor networks. In *KiVS*, pages 55–66, 2009.

[69] A. Dunkels, N. Finne, J. Eriksson, and Th. Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *SenSys*, pages 15–28. ACM Press, 2006.

[70] A. Dunkels, B. Groenvall, and Th. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN*, pages 455–462, 2004.

[71] A. Dunkels, O. Schmidt, Th. Voigt, and M. Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *SenSys*, pages 29–42. ACM Press, 2006.

[72] U. Fayyad, G. Piatetsky-shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996.

[73] C. Franke, M. Karnstedt, D. Klan, M. Gertz, K.-U. Sattler, and E. Chervakova. In-network detection of anomaly regions in sensor networks with obstacles. *Computer Science - Research and Development, Special issue "BTW 2009 Best Papers"*, 2009.

[74] C. Franke, M. Karnstedt, D. Klan, M. Gertz, K.-U. Sattler, and W. Kattanek. In-network detection of anomaly regions in sensor networks with obstacles. In *BTW*, pages 367–386, 2009.

[75] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Computer Graphics*, pages 124–133, 1980.

[76] T. Fukuda, Y. Morimoto, Sh. Morishita, and T. Tokuyama. Mining optimized association rules for numeric attributes. In *PODS 1996*, pages 182–191, 1996.

[77] M. M. Gaber, Sh. Krishnaswamy, and A. Zaslavsky. Ubiquitous data stream mining. In *Research and future directions workshop proceedings held in conjunction with the KDD*, 2004.

[78] M. M. Gaber, A. Zaslavsky, and Sh. Krishnaswamy. Mining data streams: a review. *SIGMOD Rec.*, 34(2):18–26, 2005.

[79] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[80] E. Jr. Gardner. Exponential smoothing: The state of the art–part ii. *International Journal of Forecasting*, 22(4):637–666, 2006.

[81] B. Gedik, H. Andrade, K.-L. Wu, Ph. S. Yu, and M. Doo. Spade: the system s declarative stream processing engine. In *SIGMOD*, pages 1123–1134, New York, NY, USA, 2008. ACM.

[82] B. Gedik and L. Liu. MobiEyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *EDBT*, pages 67–87, 2004.

[83] A. K. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Ssym*, pages 12–12, Berkeley, CA, USA, 1999. USENIX Association.

[84] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining Frequent Patterns in Data Streams at Multiple Time Granularities. In *Workshop on Next Generation Data Mining*, 2003.

[85] C. Giannella, J. Han, E. Robertson, and C. Liu. Mining Frequent Itemsets over Arbitrary Time Intervals in Data Streams. Technical report, Indiana University, 2003.

[86] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing Wavelets on Streams: One-Pass Summaries for Approximate Aggregate Queries. In *VLDB*, pages 79–88, 2001.

[87] L. Girod, Y. Mei, R. Newton, St. Rost, A. Thiagarajan, H. Balakrishnan, and Samuel Madden. The case for a signal-oriented data stream management system. In *CIDR*, 2007.

[88] L.s Girod, Y. Mei, St. Rost, A. Thiagarajan, H. Balakrishnan, and S. Madden. XStream: a Signal-Oriented Data Stream Management System. In *ICDE*, Cancun, Mexico, April 2008.

[89] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *The VLDB Journal*, 16(1):5–28, 2007.

[90] L. Golab, K. G. Bijay, and M. T. Özsu. Multi-query optimization of sliding window aggregates by schedule synchronization, 2006.

[91] M. Grinev and M. Pleshachkov. Rewriting-based optimization for xquery transformational queries. pages 163 – 174, jul. 2005.

[92] L. Gu and J. A. Stankovic. t-kernel: providing reliable os support to wireless sensor networks. In *Sensys '06*, pages 1–14, New York, NY, USA, 2006. ACM.

[93] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *STOC*, pages 471–475, New York, NY, USA, 2001. ACM Press.

[94] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. *TKDE*, 15(3):515–528, 2003.

[95] O. Günther. Der Zellbaum: Ein Index für geometrische Datenbanken [The cell tree: an index for geometric databases]. *Informatik - Forschung und Entwicklung*, 4(1):1–13, 1989.

[96] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, pages 47–57, 1984.

[97] Stefan Hagedorn. Raumbezogene Anfragen in Datenstrommanagementsystemen. Master's thesis, 2010.

[98] M. Halatchev and Le Gruenwald. Estimating missing values in related sensor data streams. In *COMAD*, pages 83–94, 2005.

[99] M. A. Hammad, M. F. Mokbel, M. H. Ali, W. G. Aref, A. C. Catlin, A. K. Elmagarmid, M. Eltabakh, M. G. Elfeky, T. M. Ghanem, R. Gwadera, I. F. Ilyas, M. Marzouk, and X. Xiong. Nile: A query processing engine for data streams. In *ICDE*, page 851, 2004.

[100] Ch.-Ch. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A dynamic operating system for sensor nodes. In *Mobisys*, pages 163–176, New York, NY, USA, 2005. ACM.

[101] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *SIGMOD 2000*, pages 1–12, 2000.

[102] D. J. Hand. Statistics and data mining: Intersecting disciplines. *SIGKDD Explorations*, 1:16–19, 1999.

[103] J. Hartigan. *Clustering Algorithms*. John Wiley and Sons, New York, 1975.

[104] C. Hartung, C. Seielstad, S. Holbrook, and R. Han. Firewxnet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Mobisys*, pages 28–41. ACM Press, 2006.

[105] W. R. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Mobicom*, pages 174–185, New York, NY, USA, 1999. ACM.

[106] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *In architectural support for programming languages and operating systems*, pages 93–104, New York, NY, USA, 2000.

[107] Th. Hillebrandt. Diploma-thesis: Untersuchung und simulation des zeit- und energieverhaltens eines msb430-h sensornetzwerkes. Department of Mathematics and Computer Science, FU Berlin, 2007.

[108] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:2004, 2004.

[109] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *SIGMOD*, pages 479–490, 2005.

[110] St. Ihling. Kostenbasierte Optimierung von Datenstromanfragen, 2007.

[111] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Mobicom*, pages 56–67, New York, NY, USA, 2000. ACM.

[112] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *SIGMOD*, pages 233–244, New York, NY, USA, 1995. ACM.

[113] Z. G. Ives, A. Deshpande, and V. Raman. Adaptive query processing: Why, how, when, and what next? In *VLDB*, pages 1426–1427, 2007.

[114] G. S. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *VLDB*, pages 512–523. VLDB Endowment, 2003.

[115] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review, 1999.

[116] M. Jarke. Common subexpression isolation in multiple query optimization. In *Query Processing in Database Systems*, pages 191–205. Springer, 1985.

[117] Ch. S. Jensen, D. Lin, B. Chin, and O. R. Zhang. Effective density queries on continuously moving objects. In *ICDE*, page 71, 2006.

[118] J. Jeong. Incremental network programming for wireless sensors. In *IEEE SECON*, pages 25–33, 2004.

[119] R. Jin and G. Agrawal. Efficient decision tree construction on streaming data. In *KDD*, pages 571–576, 2003.

[120] N. Kabra and D. J. DeWitt. Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. In *SIGMOD*, pages 106–117. ACM Press, 1998.

[121] H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. Vedas: A mobile and distributed data stream mining system for real-time vehicle monitoring. In Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn, editors, *SDM*. SIAM, 2004.

[122] H. Kargupta, B.-H. Park, S. Pittie, L. Liu, D. Kushraj, and K. Sarkar. Mobimine: Monitoring the stock market from a pda. *ACM SIGKDD Explorations*, 3:37–46, 2001.

[123] M. Karnstedt, D. Klan, Chr. Pölitz, K.-U. Sattler, and C. Franke. Adaptive burst detection in a stream engine. In *SAC*, pages 1511–1515, New York, NY, USA, 2009. ACM.

[124] E. Keogh, St. Lonardi, and B. Chiu. Finding surprising patterns in a time series database in linear time and space. In *KDD*, pages 550–556, 2002.

[125] D. Klan, K. Hose, M. Karnstedt, and K. Sattler. Power-Aware Data Analysis in Sensor Networks. In *ICDE 2010*, pages 1125–1128, 2010.

[126] D. Klan, K. Hose, and K.-U. Sattler. Developing and deploying sensor network applications with anduin. In *DMSN*, pages 1–6, New York, NY, USA, 2009. ACM.

[127] D. Klan, M. Karnstedt, Ch. Pölitz, and K. Sattler. Towards Burst Detection for Non-Stationary Stream Data. In *KDML*, pages 57–60, 2008.

[128] D. Klan, Th. Rohe, and K. Sattler. Quantitatives Frequent-Pattern Mining über Datenströmen. In *KDML 2010*, 2010.

[129] J. M. Kleinberg. Bursty and hierarchical structure in streams. *Data Min. Knowl. Discov.*, 7(4):373–397, 2003.

[130] E. M. Knorr and R. T. Ng. Finding intensional knowledge of distance-based outliers. In *VLDB*, pages 211–222, 1999.

[131] E. M. Knorr, R. T. Ng, and V. Tucakov. Distance-based outliers: algorithms and applications. *The VLDB Journal*, 8(3-4):237–253, 2000.

[132] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB'02*, pages 275–286, 2002.

[133] J. Krämer and B. Seeger. PIPES - A Public Infrastructure for Processing and Exploring Streams. In *SIGMOD*, pages 925–926, 2004.

[134] J. Krämer and B. Seeger. A temporal foundation for continuous queries over data streams. In *COMAD*, pages 70–82, 2005.

[135] B. Krishnamachari, D. Estrin, and St. B. Wicker. The impact of data aggregation in wireless sensor networks. In *ICDCSW*, pages 575–578, Washington, DC, USA, 2002. IEEE Computer Society.

[136] R. Kuntschke and A. Kemper. Data stream sharing. In *EDBT Workshop*, 2006.

[137] W. Labio and H. Garcia-Molina. Efficient snapshot differential algorithms in data warehousing. In *VLDB*, 1996.

[138] Y.-N. Law, H. Wang, and C. Zaniolo. Query languages and data models for database sequences and data streams. In *VLDB*, pages 492–503. VLDB Endowment, 2004.

[139] W. Lee, S. J. Stolfo, and Ph. K. Chan. Learning patterns from unix process execution traces for intrusion detection. In *AAAI workshop*, pages 50–56. AAAI Press, 1997.

[140] C. K.-S. Leung and Q. I. Khan. DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams. In *ICDM 2006*, pages 928–932, 2006.

[141] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *Sensys*, pages 126–137, New York, NY, USA, 2003. ACM.

[142] H.-F. Li, S.-Y. Lee, and M.-K. Shan. An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams. In *International workshop on knowledge discovery in data streams ICW PKDD*, Pisa, Italy, 2004.

[143] W. Lindner, H. Velke, and K. Meyer-Wegener. Data stream query optimization across system boundaries of server and sensor network. In *MDM*, page 25, Washington, DC, USA, 2006. IEEE Computer Society.

[144] St. Lindsey, C. Raghavendra, and K. M. Sivalingam. Data gathering algorithms in sensor networks using energy metrics. *IEEE Trans. Parallel Distrib. Syst.*, 13(9):924–935, 2002.

[145] Y. Liu, A. N. Choudhary, J. Zhou, and A. A. Khokhar. A scalable distributed stream mining system for highway traffic data. In *PKDD*, volume 4213 of *Lecture Notes in Computer Science*, pages 309–321. Springer, 2006.

[146] S. Madden. The design and evaluation of a query processing architecture for sensor networks. Technical report, 2003.

[147] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[148] S. Madden and M.J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *ICDE*, pages 555–, Washington, DC, USA, 2002. IEEE Computer Society.

[149] S .R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Trans. on Database Systems*, 30(1):122–173, 2005.

[150] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA*, pages 88–97. ACM, 2002.

[151] S. Maneewongvatana and D. M. Mount. It's okay to be skinny, if your friends are fat. In *Center for Geometric Computing 4th Annual Workshop on Computational Geometry*, 1999.

[152] G. S. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *VLDB 2002*, pages 346–357, 2002.

[153] A. Mann. Multi-Query-Optimierung von Datenstromanfragen. DBIS, TU Ilmenau, Diplomarbeit, 2007.

[154] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963.

[155] P. J Marron, M. Gauger, A. Lachenmann, D. Minder, O. Saukh, and K. Rothermel. Flexcup: A flexible and efficient code update mechanism for sensor networks. In *EWSN 2006*, pages 212–227, 2006.

[156] J. Mata, J.L. Alvarez, and J.C. Riquelme. An Evolutionary Algorithm to Discover Numeric Association Rules. In *SAC 2002*, pages 590–594, 2002.

[157] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *VLDB*, pages 101–110. Morgan Kaufmann Publishers Inc., 2000.

[158] Anand Meka. Distributed spatial clustering in sensor networks. In *EDBT*, 2006.

[159] J. Melton and A. Eisenberg. Sql multimedia and application packages (sql/mm). *SIGMOD Rec.*, 30(4):97–102, 2001.

[160] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. Yale: Rapid prototyping for complex data mining tasks. In *KDD*, pages 935–940, New York, NY, USA, 2006. ACM.

[161] A. Milenkovic, Ch. Otto, and E. Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications*, 29:2521–2533, 2006.

[162] R. Miller and S. K. McKenzie. Time series modeling of monthly general fertility rates, 1984.

[163] M. F. Mokbel and W. G. Aref. PLACE: A scalable location-aware database server for spatio-temporal data streams. *IEEE Data Eng. Bull.*, 28(3):3–10, 2005.

[164] M. F. Mokbel, X. Xiong, M. A. Hammad, and W. G. Aref. Continuous query processing of spatio-temporal data streams in PLACE. *Geoinformatica*, 9(4):343–365, 2005.

[165] L. Mottola and G. P. Picco. Programming wireless sensor networks: Fundamental concepts and state-of-the-art. Politecnico di Milano, Milano, Italy.

[166] L. Mottola and G. P. Picco. Logical neighborhoods: a programming abstraction for Wireless Sensor Networks. In *DCOSS*, 2006.

[167] M. Muralikrishna and David J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *SIGMOD*, pages 28–36, 1988.

[168] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.

[169] E. Özkural and C. Aykanat. A space optimization for fp-growth. In *FIMI*, 2004.

[170] P. Chandrakasan and W.B. Heinzelman. Application-specific protocol architectures for wireless networks. *IEEE Transactions on Wireless Communications*, 1:660–670, 2000.

[171] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD'03*, pages 467–478, 2003.

[172] J. S. Park, M.-S. Chen, and P. S. Yu. An Effective Hash-based Algorithm for Mining Association Rules. In *SIGMOD '95*, pages 175–186, 1995.

[173] G. Piatetsky-Shapiro, R. J. Brachman, T. Khabaza, W. Klösgen, and E. Simoudis. An overview of issues in developing industrial data mining and knowledge discovery applications. In *KDD*, pages 89–95, 1996.

[174] Ch. Pölitz. Burstdetection auf Datenströmen, 2008. Studienarbeit.

[175] F. P. Preparata and M. I. Shamos. *Computational Geometry - An Introduction*. Springer, 1985.

[176] R. Rajagopalan and P. K. Varshney. Data aggregation techniques in sensor networks: A survey. *Comm. Surveys & Tutorials, IEEE*, 8:48–63, 2006.

[177] V. Raman, B. Raman, and J. M. Hellerstein. Online dynamic reordering. *The VLDB Journal*, 9(3):247–260, 2000.

[178] Th. Rohe. Energieeffiziente Erkennung von Frequent Pattern in Sensornetzen. Master's thesis, TU Ilmenau, 2010.

[179] K. Römer. Discovery of Frequent Distributed Event Patterns in Sensor Networks. In *EWSN 2008*, pages 106–124, 2008.

[180] E. A. Rundensteiner, L. Ding, T. Sutherland, Y. Zhu, B. Pielech, and N. Mehta. CAPE: Continuous query engine with heterogeneous-grained adaptivity. In *VLDB*, pages 1353–1356, Toronto, Canada, 2004.

[181] G. Saake, A. Heuer, and K.Sattler. *Datenbanken: Implementierungstechniken, 2. Auflage*. mitp-Verlag, Januar 2005.

[182] A. Salleb-Aouissi, C. Vrain, C., and Nortet. Quantminer: a genetic algorithm for mining quantitative association rules. In *IJCAI 2007*, pages 1035–1040, 2007.

[183] S. Saltenis, Ch. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects, 2000.

[184] K. Sattler, O. Dunemann, I. Geist, G. Saake, and S. Conrad. Limiting Result Cardinalities for Multidatabase Queries using Histograms. In *BNCOD*, volume 2097 of *LNCS*, pages 152–167, Oxford, U.K., 2001.

[185] S. Schmidt, H. Berthold, and W. Lehner. Qstream: Deterministic querying of data streams. In *VLDB*, pages 1365–1368, 2004.

[186] S. Schmidt, Th. Legler, S. Schär, and W. Lehner. Robust real-time query processing with qstream. In *VLDB*, pages 1299–1301. VLDB Endowment, 2005.

[187] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *IEEE symposium on security and privacy*, pages 144–155, 2001.

[188] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In *The VLDB Journal*, pages 507–518, 1987.

[189] D. Shasha and Y. Zhu. *High Performance Discovery in Time Series: Techniques and Case Studies*. Springer, 2004.

[190] K. Shim, T. Sellis, and D. Nau. Improvements on a heuristic algorithm for multiple-query optimization. *Data Knowl. Eng.*, 12(2):197–222, 1994.

[191] A. Silberstein, R. Braynard, and J. Yang. Constraint chaining: on energy-efficient continuous monitoring in sensor networks. In *SIGMOD*, pages 157–168, New York, NY, USA, 2006. ACM.

[192] A. Soheili, V. Kalogeraki, and D. Gunopulos. Spatial queries in sensor networks. In *GIS*, pages 61–70, New York, NY, USA, 2005. ACM.

[193] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. *SIGMOD Rec.*, 25(2):1–12, 1996.

[194] U. Srivastava, K. Munagala, and J. Widom. Operator placement for in-network stream query processing. Technical report, Stanford InfoLab, 2004.

[195] St. Conrad. *Föderierte Datenbanksysteme: Konzepte der Datenintegration.* Habilitationsschrift, 1997.

[196] K. Stolze. Sql/mm spatial - the standard to manage spatial data in a relational database system. In *BTW*, pages 247–264, 2003.

[197] K. Stolze. *Integration of Spatial Vector Data in Enterprise Relational Database Environments.* PhD thesis, FSU Jena, Germany, PhD-Thesis, 2006.

[198] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison-Wesley, 2005.

[199] S. K. Tanbeer, Ch. F. Ahmed, B.-S. Jeong, and Y.-K. Lee. Efficient Frequent Pattern Mining over Data Streams. In *CIKM '08*, pages 1447–1448, 2008.

[200] S. K. Tanbeer, Ch. F. Ahmed, B.-S. Jeong, and Y.-K. Lee. Efficient single-pass frequent pattern mining using a prefix-tree. *Inf. Sci.*, 179(5):559–583, 2009.

[201] N. Tatbul, U. Centintemel, S. B. Zdonik, M. Cherniack, and M. Stonebrake. Load shedding a data stream manager. In *VLDB 2003*, 2003.

[202] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *DCOSS*, pages 307–321, 2005.

[203] P. Tucker, K. Tufte, V. Papadimos, and D. Maier. NEXMark - A Benchmark for Queries over Data Streams DRAFT.

[204] J. D. Ullman, H. Garcia-Molina, and J. Widom. *Database Systems: The Complete Book.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[205] U. Varshney. Patient monitoring using infrastructureoriented wireless lans. *Int. J. Electronic Healthcare*, 2:149–163, 2006.

[206] Sh. Venkataraman and T. Zhang. Heterogeneous database query optimization in db2 universal datajoiner. In *VLDB*, pages 685–689. Morgan Kaufmann, 1998.

[207] St. D. Viglas and J. F. Naughton. Rate-based query optimization for streaming information sources. In *SIGMOD*, pages 37–48, New York, NY, USA, 2002. ACM.

[208] Th. Vincenty. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, 22(176):88–93, 1975.

[209] J. S. Vitter, M. Wang, and B. R. Iyer. Data Cube Approximation and Histograms via Wavelets. In *CIKM*, pages 96–104, 1998.

[210] M. Vlachos, Ch. Meek, Z. Vagena, and D. Gunopulos. Identifying Similarities, Periodicities and Bursts for Online Search Queries. In *SIGMOD*, pages 131–142, 2004.

[211] Brett W. and Kristofer S. J. Smart dust: Communicating with a cubic-millimeter computer. *Classical Papers on Computational Logic*, 1:372–383, 2001.

[212] K. Wang, L. Tang, J. Han, and J. Liu. Top Down FP-Growth for Association Rule Mining. In *PAKDD '02*, pages 334–340, 2002.

[213] M. Wang, T. M. Madhyastha, N. H. Chan, S. Papadimitriou, and Ch. Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *ICDE*, 2002.

[214] S. Wang, E. Rundensteiner, S. Ganguly, and S. Bhatnagar. State-slice: new paradigm of multi-query optimization of window-based stream queries. In *VLDB*, pages 619–630. VLDB Endowment, 2006.

[215] J. Widom and R. Motwani. Query processing, resource management, and approximation in a data stream management system. In *CIDR*, pages 245–256, 2003.

[216] O. Wolfson. Moving objects information management: The database challenge. In *NGITS*, pages 75–89. Springer, 2002.

[217] X. Wang, J. Wang, and Y. Xu. Data Dissemination in Wireless Sensor Networks with Network Coding. *EURASIP Journal on Wireless Communications and Networking*, 2010.

[218] Sh. Xiang, H. B. Lim, and K.-L. Tan. Impact of multi-query optimization in sensor networks. In *DMSN'06*, pages 7–12, New York, NY, USA, 2006.

[219] Sh. Xiang, H.-B. Lim, and K.-L. Tan. Multiple query optimization for wireless sensor networks. In *ICDE*, 2007.

[220] Sh. Xiang, H. B. Lim, K.-L. Tan, and Y. Zhou. Two-tier multiple query optimization for sensor networks. In *ICDCS*, page 39, Washington, DC, USA, 2007. IEEE Computer Society.

[221] X. Xiong, H. G. Elmongui, X. Chai, and W. G. Aref. PLACE*: A distributed spatio-temporal data stream management system for moving objects. In *MDM*, pages 44–51, 2007.

[222] Y. Yao and J. Gehrke. Query processing for sensor networks, 2003.

[223] Y. Yao and J. Gehrke. Query processing in sensor networks. CIDR, January 2003.

[224] Y. Yao and J. E. Gehrke. The cougar approach to in-network query processing in sensor networks. *ACM SIGMOD Record*, 31(2):9–18, 2002.

[225] O. Younis and S. Fahmy. Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Transactions on Mobile Computing*, 3:366–379, 2004.

[226] St. Zdonik, M. Stonebraker, M. Cherniack, U. C. Etintemel, M. Balazinska, and H. Balakrishnan. The aurora and medusa projects. *IEEE Data Engineering Bulletin*, 26, 2003.

[227] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *SIGMOD*, pages 103–114, 1996.

[228] X. Zhang. *Fast algorithms for burst detection.* PhD thesis, New York, NY, USA, 2006. Adviser-Shasha, Dennis.

[229] Xin Zhang and Dennis Shasha. Better burst detection. In *ICDE '06*, pages 146–149, 2006.

[230] Y. Zhao, P. M. Deshpande, J. F. Naughton, and A. Shukla. Simultaneous optimization and evaluation of multiple dimensional queries. *SIGMOD Rec.*, 27(2):271–282, 1998.

[231] Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *VLDB*, 2002.

[232] Y. Zhu and D. Shasha. Efficient Elastic Burst Detection in Data Streams. In *KDD 2003, Washington, D.C., USA*, pages 336–345, 2003.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalte der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und den erfolglosen Abbruch des Promotionsverfahrens zu Folge hat.

Ilmenau, 14. März 2011

Thesen

- Das in der Arbeit vorgestellte System *AnduIN* erlaubt eine plattformübergreifende Bearbeitung von Datenstromanfragen. Anfragen können dabei sowohl zentral von einem Datenstrom-Managementsystem als auch verteilt in einem drahtlosen Sensornetzwerk ausgeführt werden.

- Der anfragespezifische Aufbau der Sensorknoten-Laufzeitumgebung ermöglicht die Integration nahezu beliebiger Operatoren.

- Das entwickelte System kann selbstständig entscheiden auf welcher Systemkomponente, welche Teile einer Anfrage ausgeführt werden.

- Mittels des in der Arbeit vorgestellten Kostenmodells können Anfragepläne bezüglich ihrer Energieeffizienz und ihres Datendurchsatzes hinreichend genau bewertet werden, um aus einer Menge physischer Anfragepläne den in der tatsächlichen Ausführung kostengünstigsten zu wählen.

- Das vorgestellte Kostenmodell erlaubt eine einheitliche plattformübergreifende Kostenabschätzung. Die Kostenbestimmung basiert dabei ausschließlich auf statistisch erfassbaren Werten wie der Input- und der Outputrate der Operatoren und den Ausführungskosten der Operatoren.

- Das in der Arbeit vorgestellte Konzept zur Zerlegung komplexer Operatoren erlaubt eine plattformübergreifende Durchführung von komplexen Data-Mining-Verfahren.

- Bei der plattformübergreifenden Optimierung von Anfragen müssen mindestens zwei Optimierungsdimensionen berücksichtigt werden. Aus dem Nutzerfeedback lassen sich passende Parameter für die automatische multidimensionale Optimierung ableiten.

- Durch das Zusammenführen von Samplingoperatoren und die Synchronisation von Samplingperioden mittels zusätzlicher Operatoren können Energiekosten im WSN eingespart werden.

- Räumliche Anfragen über Datenströmen mit Geoinformationen können im vorgestellten System effizient verarbeitet werden.

- Das vorgestellte Verfahren zur adaptiven Burst-Erkennung liefert für trendbehaftete Datenströme bessere Ergebnisse als bisher existierende Verfahren.
- Die Erkennung quantitativer Itemsets in Datenströmen ist mittels des FP^2-Stream effizient möglich. Das Verfahren ist für die Verarbeitung von Sensordatenströmen geeignet und lässt sich partiell auf den Knoten im Sensornetzwerk ausführen. Die In-Network Verarbeitung ist dabei energieeffizienter als die ausschließlich zentrale Verarbeitung im Datenstrom-Managementsystem.