

# Nutzung von Indizes auf Sensornetzen

Autor: Marc Philipp ([marc.philipp@ira.uka.de](mailto:marc.philipp@ira.uka.de))  
Betreuer: Markus Bestehorn ([bestehorn@ipd.uka.de](mailto:bestehorn@ipd.uka.de))

Seminar „Informationsverwaltung in Sensornetzen“  
Institut für Programmstrukturen und Datenorganisation  
Lehrstuhl für Systeme der Informationsverwaltung (Prof. Böhm)  
Universität Karlsruhe (TH)

**Zusammenfassung.** Sensornetze werden häufig als verteilte Datenbanken betrachtet. Diese Abstraktion führt allerdings in vielen Fällen zu Effizienzeinbußen. Gerade bei Sensornetzen will man unnötigen Aufwand vermeiden, da die Sensoren meist nur einen stark begrenzten Energievorrat besitzen. Ähnlich wie in traditionellen Datenbanksystemen wird versucht, das Problem durch vorberechnete Indexstrukturen zu lösen. Die Voraussetzungen für solche Indizes auf Sensornetzen sind jedoch grundverschieden. Daher wird nach neuen Lösungen für derartige verteilte Datenstrukturen gesucht. Diese Seminararbeit stellt drei dieser Ansätze vor und bewertet sie hinsichtlich ihrer Praxistauglichkeit.

## 1 Einleitung

Die Abstraktion eines Sensornetzes als verteilte Datenbank ist eine weit verbreitete und nützliche Sichtweise. Die so gewonnene Deklarativität von Anfragen und die Wiederverwendbarkeit von Sensornetzen führt jedoch häufig zu Einbußen bei der Effizienz. Ähnlich wie in herkömmlichen Datenbanksystemen versucht man, diese durch die Nutzung von Indexstrukturen, die vorberechnete Daten enthalten, auszugleichen. Indizes nehmen zusätzliche Kosten zur Vorbereitung in Kauf, um die Bearbeitung von Anfragen zu beschleunigen.

Allerdings unterscheiden sich die Voraussetzungen und Zielsetzungen für Indizes auf Sensornetzen grundsätzlich von denen herkömmlicher Datenbanksysteme. Sensorknoten beziehen ihre Energie aus Batterien, deren Größe dominierend für die Gesamtgröße eines Sensors ist. Die Lebenszeit eines Sensors hängt hauptsächlich davon ab, wie schnell seine Energie erschöpft ist. Wichtigstes Entwurfsziel für Indizes auf Sensornetzen ist also die Energieeffizienz. Die Kosten für die Kommunikation sind dabei deutlich höher als die Kosten für lokale Berechnungen.

An Indizes auf Sensornetzen wird eine weitere Anforderung gestellt: Sie müssen im Netz selbst aufgebaut, gespeichert und verwaltet werden. Eine Alternative besteht darin, einen Sensor als zentralen Datenbankknoten auszuzeichnen, dem alle anderen Sensoren regelmäßig ihre Daten schicken. Dieser Knoten könnte dann herkömmliche Datenbanktechnologie benutzen, um die Daten zu verwalten.

Allerdings ist dies nicht praktikabel, da Sensoren, die nahe an dem Datenbankknoten liegen, zu einem Flaschenhals werden und ihre Energie schnell erschöpft ist.

Eine weitere Möglichkeit, ohne einen im Netzwerk gehaltenen Index auszukommen, ist das Fluten von Anfragen. In einem solchen Schema speichern Sensoren ausschließlich ihre eigenen Daten. Eine Anfrage wird an alle Sensoren geschickt (*Flooding*), die daraufhin eine Antwort an den Absender zurücksenden. Auch diese Möglichkeit ist aus Gründen der Energieeffizienz – zumindest in dieser naiven Ausführung – nicht zu gebrauchen.

Neben den Kosten für die Bearbeitung von Anfragen dürfen die Kosten für Aktualisierungen des Index nicht vernachlässigt werden. Sensoren liefern häufig einen Strom von Daten, d. h. es müssen laufend Updates verarbeitet werden.

Im folgenden Abschnitt werden zunächst Bereichsanfragen eingeführt und motiviert. Danach werden drei Ansätze für verteilte Indexstrukturen auf Sensornetzen vorgestellt und bewertet. Als erstes Verfahren werden „*Distributed Indices for Multi-dimensional Data*“ (DIMs) [1] behandelt, danach folgen „*Fractionally Cascaded Information in a Sensor Network*“ [2] und schließlich „*Sensor Trees*“ [3].

## 2 Bereichsanfragen

Bereichsanfragen bieten eine natürliche Möglichkeit, ein Informationsbedürfnis zu formulieren. Dies ist zum einen notwendig, da Messgeräte ungenau sind, also nur ungefähre Werte liefern. Zum anderen ist das Informationsbedürfnis eines Benutzers häufig unscharf, d. h. er interessiert sich für alle Werte in einem bestimmten Intervall und nicht für einen konkreten Messwert.

Mehr-dimensionale Bereichsanfragen bieten zusätzlich die Möglichkeit, die Korrelation von Daten auszunutzen. Beispielsweise könnte man in einem Szenario zur Lebensraumüberwachung wie Great Duck Island [4] bei bestimmten Messwerten einer Kamera und eines Mikrofons einen Wissenschaftler benachrichtigen; jedoch nicht, wenn nur einer der beiden Sensoren einen der gesuchten Werte liefert.

*Beispiel 1.* Es sei ein Sensornetz gegeben, in dem die Sensoren sowohl die Temperatur als auch die Luftfeuchtigkeit messen. Ein Beispiel einer mehr-dimensionalen Bereichsanfrage ist „Liste alle Messungen mit Temperaturen zwischen 20°C und 30°C und Luftfeuchtigkeit von über 50%“ bzw. als Ausdruck in einer SQL-artigen Anfragesprache:

```
SELECT temperature, humidity
FROM   sensors
WHERE  temperature >= 20
AND    temperature <= 30
AND    humidity > 0.5
```

### 3 Verteilte Indizes für mehrdimensionale Daten

Der erste Ansatz „*Distributed Indices for Multi-dimensional Data*“ (DIMs) [1] versucht, die Abarbeitung mehr-dimensionaler Bereichsanfragen effizienter zu gestalten. DIMs sind eine verteilte Datenstruktur, die auf dem Prinzip beruht, dass jeder Sensor für einen gewissen Bereich der Daten zuständig ist, d. h. ausschließlich diese Daten speichert. Das Sensorfeld wird in Zonen aufgeteilt, anfallende Daten werden einer Zone zugeordnet und dort gespeichert. Dazu benutzen DIMs ähnlich wie geographische Hashtabellen (GHT) [5] eine Hashfunktion.

DIMs machen eine Reihe von Annahmen, die im Folgenden benutzt werden. Zum einen muss jeder Sensorknoten seine Position sowie die der Netzwerk Grenzen kennen und zudem fest installiert sein. Außerdem wird angenommen, dass alle Attributwerte normalisiert sind, d. h. zwischen 0 und 1 liegen. Dies ist allerdings keine wirkliche Einschränkung, da dies leicht durch eine Abbildung des tatsächlich vorkommenden Intervalls auf  $[0, 1]$  realisiert werden kann.

Das Verfahren basiert auf dem Routingprotokoll *Greedy Perimeter Stateless Routing* (GPSR) [6], das im folgenden Abschnitt erläutert wird.

#### 3.1 Greedy Perimeter Stateless Routing

GPSR ist ein geographisches Routingverfahren, d. h. es ermöglicht die Zustellung eines Pakets bzw. einer Nachricht an einen Sensorknoten, der sich an einer bestimmten Position befindet. Adressat ist hierbei nicht ein bestimmter Sensor, sondern ein geographischer Ort. Eine Nachricht wird demjenigen Sensorknoten zugestellt, der dem Ziel am nächsten ist.

Das Verfahren kommt nahezu ohne die Verwaltung von Routingtabellen etc. aus und bezeichnet sich daher als zustandslos (*stateless*). Es werden lediglich Informationen über direkte Nachbarn gespeichert.

Pakete werden von GPSR in zwei verschiedenen Modi zwischen den einzelnen Sensoren weitergeleitet. Wo möglich, wird der *Greedy*-Modus verwendet, unter Umständen muss jedoch auf den *Perimeter*-Modus zurückgegriffen werden.

Jeder Knoten, der ein Datenpaket erhält, überprüft zunächst, ob er selbst der Empfänger ist. Das ist entweder dann der Fall, wenn die Koordinaten des Knotens exakt mit den Zielkoordinaten des Pakets übereinstimmen oder die Reichweite des Knotens so groß ist, dass er sicher sein kann, dass sich kein anderer Knoten näher am Ziel befindet.

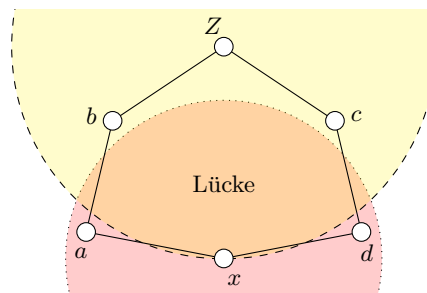
Kennt der Knoten einen Nachbarn, der dem Ziel näher ist, leitet er das Datenpaket im *Greedy*-Modus an diesen weiter. Ist das nicht der Fall, kommt der *Perimeter*-Modus zum Einsatz (siehe Abb. 1). Dazu vermerkt der Knoten zunächst seine eigenen Koordinaten im Paket, um es später erkennen zu können, falls es an ihn zurückgeschickt werden sollte. Dies geschieht dann, wenn sich kein Knoten findet, der näher am Ziel ist.

Die Auswahl des nächsten Hops im *Perimeter*-Modus ist durch die sogenannte *Right-Hand Rule* festgelegt: Der Knoten schickt das Paket an den ersten Nachbarn, auf den er trifft, wenn er sich, in Zielrichtung schauend, entgegen dem Uhrzeigersinn dreht. Voraussetzung für die korrekte Funktionsweise von

GPSR ist, dass das Netzwerk als planarer Graph dargestellt werden kann. Diese Eigenschaft muss vor Verwendung von GPSR sichergestellt sein.

*Beispiel 2.* Im Szenario, das in Abb. 1 dargestellt wird, erhält  $x$  ein Datenpaket mit Ziel  $Z$ .  $x$  ist das lokale Minimum bezüglich der Entfernung zu  $Z$ . Weil eine Weiterleitung im *Greedy*-Modus nicht möglich ist, sendet  $x$  nach Anwendung der *Right-Hand Rule* das Paket im *Perimeter*-Modus an  $a$  weiter.

Der Sensorknoten  $a$  kennt nun einen Knoten  $b$ , der näher an  $Z$  liegt. Folglich gelangt das Paket im *Greedy*-Modus zu  $b$  und von dort schließlich zu  $Z$ .



**Abb. 1.** *Perimeter*-Modus zur Umgehung einer Lücke

### 3.2 Aufteilung in Zonen

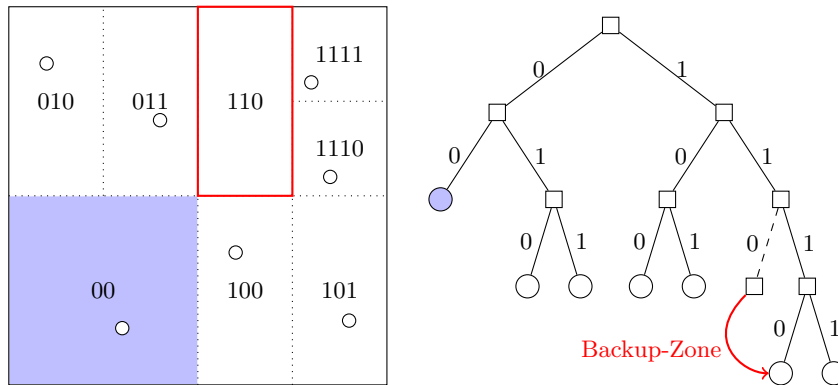
Zunächst wird das rechteckige Sensorfeld  $R$  abwechselnd vertikal und horizontal in gleichgroße Hälften geteilt, bis in jeder Zone nur noch ein Sensor liegt. Dabei wird jeder Zone  $A$  ein eindeutiger Code  $Z_A$  zugeordnet:

- Falls  $A$  in der linken Hälfte von  $R$  liegt, ist das erste Bit 0, sonst 1.
- Falls  $A$  in der unteren Hälfte von  $R$  liegt, ist das zweite Bit 0, sonst 1.

Diese Prozedur wird nun rekursiv für alle Teilzonen fortgesetzt. Die Aufteilung in Zonen kann durch einen virtuellen Binärbaum repräsentiert werden. Der Baum wird als Datenstruktur allerdings nicht gespeichert, sondern ist nur implizit durch die Sensorknoten vorhanden.

*Beispiel 3.* In Abb. 2 ist eine solche Aufteilung für ein Sensornetz mit 7 Knoten angegeben. Jeder Blattknoten im Baum entspricht einer Zone, z. B. ist Zone 00 der Knoten ganz links im Binärbaum.

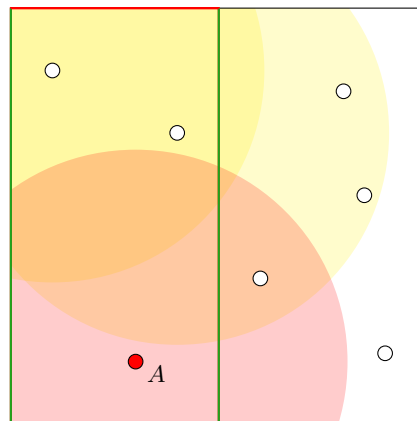
Konkret läuft der Algorithmus allerdings verteilt ab: Jeder Sensor versucht, seine eigene Zone zu bestimmen. Dabei kommuniziert er ausschließlich mit seinen direkten Nachbarn, d. h. den Sensorknoten innerhalb seiner Reichweite. Ein Sensorknoten geht zunächst davon aus, das gesamte Sensorfeld als Zone zu besitzen. Seine Zonengrenzen werden dabei als *unbestimmt* markiert. Überlappt sich



**Abb. 2.** Aufteilung eines Sensorfelds in Zonen

seine Reichweite in einer Richtung mit einer Netzwerkgrenze, wird diese Grenze als *bestimmt* markiert. Bei erfolgreicher Kommunikation mit einem Nachbarn verkleinert ein Knoten seine Zone nach obigem Verfahren solange, bis sich der Nachbarknoten nicht mehr in seiner Zone befindet. Falls in einer Richtung keine Kommunikation möglich ist, verbleiben die entsprechenden Zonengrenzen zunächst mit *unbestimmter* Markierung und werden erst bei der Bearbeitung einer Anfrage oder dem Einfügen eines Datums aufgelöst.

*Beispiel 4.* In Abb. 3 ist ein Beispielszenario gegeben, in dem ein Sensorknoten *A* aufgrund zu geringer Reichweite von einer zu großen Zone ausgeht. Die obere Grenze seiner Zone wird als *unbestimmt* markiert.



**Abb. 3.** Unbestimmte Zonengrenzen

Ein weiteres Problem des Aufteilungsprozesses besteht darin, dass Zonen leer bleiben können. Deshalb wird jeder Zone eine Backup-Zone zugeteilt. Bleibt eine Zone leer, werden anfallende Daten in der zugehörigen Backup-Zone gespeichert. Die Backup-Zone ist der Knoten im virtuellen Binärbaum, der sich am weitesten links (bzw. rechts) im Teilbaum des rechten (bzw. linken) Bruders befindet. Die Konstruktion stellt sicher, dass eine Zone stets an ihre Backup-Zone grenzt.

*Beispiel 5.* In Abb. 2 ist Zone 110 leer geblieben und bekommt 1110 als Backup-Zone zugewiesen.

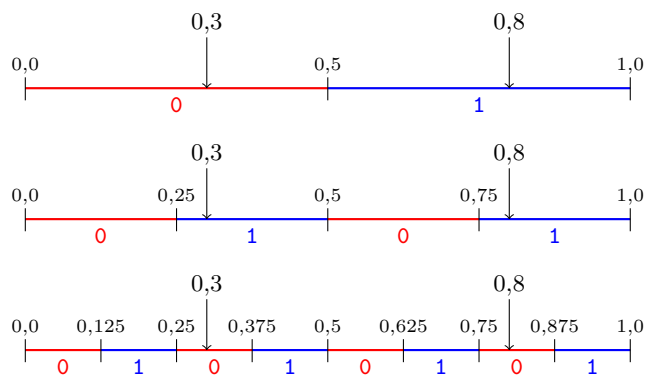
### 3.3 Abbildung eines Datums auf einen Ort

Die Zuordnung eines Datums  $D = \langle A_1, \dots, A_m \rangle$  zu einem Ort geschieht über eine Hashfunktion. Dazu wird der normalisierte Wertebereich  $[0, 1]$  der Attribute sukzessiv in gleichgroße Intervalle unterteilt, die jeweils mit 0 bzw. 1 kodiert werden. Ein Knoten kodiert dabei so lange, bis der Code die Länge seines eigenen Zonencodes hat. Die Hashfunktion ist lokali-täts-erhaltend, d. h. Daten mit ähnlichen Attributwerten werden auf ähnliche Funktionswerte abgebildet.

*Beispiel 6.* Es sei ein Knoten  $A$  mit einem Zonencode  $Z_A$  der Länge 5 gegeben.  $A$  soll die Hashfunktion für das Datum  $\langle 0,3; 0,8 \rangle$  berechnen (siehe Abb. 4).

Dazu wird das Intervall  $[0, 1]$  zunächst in die Teilintervalle  $I_1 = [0; 0,5]$  und  $I_2 = [0,5; 1]$  aufgeteilt. Der erste Wert des Datums 0,3 liegt in  $I_1$  und wird daher zu 0 kodiert, der zweite Wert 0,8 liegt in  $I_2$  und wird zu 1. Als Zwischenergebnis hat  $A$  also 01 errechnet.

Die bisherigen Intervalle  $I_1$  und  $I_2$  werden nun erneut in jeweils zwei Teilintervalle aufgeteilt, die abwechselnd mit 0 und 1 kodiert werden. Sobald das Zwischenergebnis die Länge 5 hat, bricht die Berechnung ab. Das Endergebnis der Hashfunktion ist in diesem Fall 01110.



**Abb. 4.** Knoten  $A$  mit Zonencode  $Z_A$  der Länge 5 kodiert  $\langle 0,3; 0,8 \rangle \mapsto 01110$

Nach der Kodierung des Datums im Ursprungsknoten kommt GPSR zum Einsatz, um das Datum seinem Besitzer zuzustellen. Der Besitzer eines Datums ist derjenige Sensorknoten, dessen Zonencode den längsten gemeinsamen Präfix mit dem Code des Datums aufweist.

Initial vergleicht der Ursprungsknoten  $A$  seinen Zonencode  $Z_A$  mit dem errechneten Code des Datums  $\text{code}_A(D)$ . Sind die beiden Werte identisch, so ist  $A$  der Besitzer und speichert das Datum. Ansonsten markiert sich  $A$  als initialer Besitzer des Datums ( $\text{owner}(D)$ ) und sendet per GPSR eine Nachricht, bestehend aus

$$\langle D, \text{code}(D), \text{owner}(D) \rangle,$$

an den Mittelpunkt der Zone  $Z' = \text{code}(D)$ .

Jeder Knoten  $B$  auf dem Weg von  $A$  zum Besitzer von  $D$  berechnet erneut den Code des Datums  $\text{code}_B(D)$ . Falls der neue Code länger ist, aktualisiert  $B$  den in der Nachricht enthaltenen Code  $\text{code}(D)$ . Der Adressat wird so schrittweise verfeinert. Dies ist notwendig, da ein einzelner Sensorknoten eine eingeschränkte Sicht auf das Netzwerk hat und nicht weiß, wie lang der längste vorkommende Zonencode ist.

$B$  vergleicht nun  $Z_B$  mit  $\text{owner}(D)$ . Falls  $Z_B$  einen längeren gemeinsamen Präfix mit  $\text{code}(D)$  hat als  $\text{owner}(D)$ , setzt  $B$  sich selbst als neuen Besitzer von  $D$ . Falls der Zonencode  $Z_B$  exakt mit  $\text{code}(D)$  übereinstimmt, speichert  $B$  das Datum; ansonsten sendet  $B$  eine Nachricht an den Mittelpunkt der Zone  $Z'' = \text{code}(D)$ . GPSR garantiert, dass die Nachricht an  $B$  zurückgeschickt wird, sollte sich kein Knoten finden, der besser geeignet ist.

*Beispiel 7.* Wie dabei nebenher *unbestimmte* Zonengrenzen aufgelöst werden, sieht man in Abb. 5. Der Sensor  $A$  erhält die Daten  $\langle 0,3; 0,8 \rangle$  und kodiert diese zu 0. Da eine Grenze von  $A$  *unbestimmt* ist, markiert sich  $A$  im Paket als vorläufiger Besitzer, leitet es aber an  $B$  weiter.  $B$  berechnet 011 und leitet das Paket daher an  $C$  weiter.  $C$  erkennt sich selbst als Besitzer und speichert die Daten. Außerdem entdeckt  $C$  anhand des in der Nachricht angegebenen Besitzers 0, dass  $A$  von einer zu großen Zone ausgeht. Also sendet  $C$  eine Verkleinerungsnachricht (*shrink message*) an  $A$ , woraufhin  $A$  seine Zone zu 00 korrigiert und die zugehörige Zonengrenze als *bestimmt* markiert.

### 3.4 Bearbeitung von Anfragen

DIMs sehen vor, dass jeder Sensorknoten eine Anfrage absetzen kann. Zur Verarbeitung wird die Anfrage zunächst zu einem Sensorknoten in der kleinsten Zone geroutet, die den gesamten Wertebereich der Anfrage abdeckt. Dieser Knoten hat möglicherweise eine genauere Sicht auf das Netz und wird die Anfrage weiterleiten, bis sie bei einem Zwischenknoten eintrifft, dessen Zone tatsächlich eine Überlappung mit dem angefragten Wertebereich aufweist. Dieser Knoten teilt die Anfrage nun in Teilanfragen auf, die wiederum an die entsprechenden Zonen verschickt werden. Nach dem Eintreffen der Teilergebnisse beim Zwischenknoten wird dort das Gesamtergebnis zusammengesetzt und an den Knoten zurückgesandt, der die Anfrage initiiert hat.

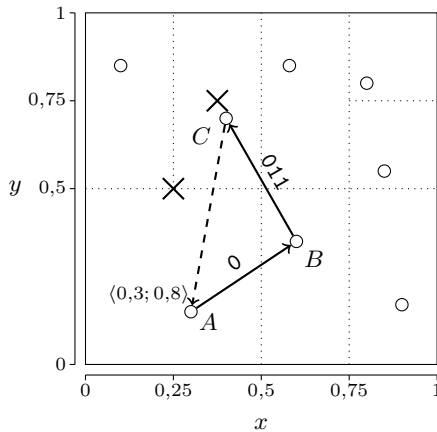


Abb. 5. Routing zum Besitzer

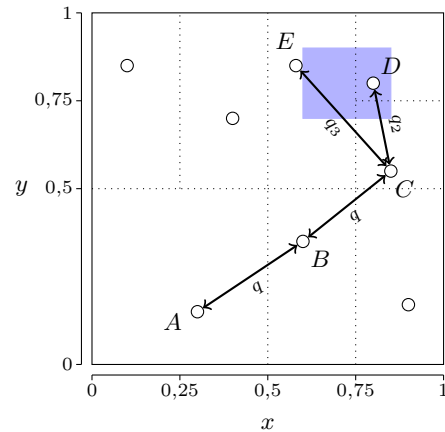


Abb. 6. Bearbeitung einer Anfrage

*Beispiel 8.* In Abb. 6 wird die Anfrage  $q = \langle 0,6 - 0,85; 0,7 - 0,9 \rangle$  von A abgesetzt. Aus Sicht von A ist Zone 1 zuständig, also B. B hat eine detailliertere Sicht auf das Sensornetz und leitet  $q$  daher an C weiter. C teilt  $q$  in drei Teilanfragen auf. C selbst bearbeitet  $q_1 = \langle 0,75 - 0,85; 0,7 - 0,75 \rangle$ , D bzw. E erhalten die Anfragen  $q_2 = \langle 0,75 - 0,85; 0,75 - 0,9 \rangle$  bzw.  $q_3 = \langle 0,6 - 0,75; 0,7 - 0,9 \rangle$ . Nach Abschluss der Bearbeitung senden D und E ihre Teilergebnisse an C zurück. C setzt das Ergebnis aus den Teilergebnissen zusammen und leitet es über B an A zurück.

### 3.5 Bewertung des Verfahrens

DIMs benutzen eine Hashfunktion zur Indizierung, die Lokalität bezüglich der Daten gewährleistet. Dabei geht jedoch die räumliche Lokalität der Daten verloren, d. h. Anfragen, die sich nur auf gewisse Orte beziehen, werden nicht unterstützt. Dies lässt sich dadurch beheben, dass die Koordinaten  $(x, y)$  der Sensoren als zusätzliche Attribute aufgefasst werden und mit in die Berechnung der Hashfunktion eingehen.

Eine weitere Einschränkung besteht hinsichtlich der Verteilung der Daten. Falls die Attributwerte in einer oder mehreren Dimensionen nicht gleichverteilt sind, müssen Sensorknoten, die für eines der häufiger vorkommenden Teilintervalle zuständig sind, unter Umständen weitaus mehr Daten speichern als andere Sensoren. Dadurch benötigen sie auch deutlich mehr Energie zur Kommunikation und fallen früher aus.

Einen weiteren Kritikpunkt stellt die Verarbeitung von Updates dar. Im Gegensatz zu einem Ansatz, bei dem Sensoren nur ihre eigenen Daten speichern und Anfragen an das gesamte Netzwerk geflutet werden, entstehen bei DIMs für jede Veränderung der Werte eines Sensorknotens zusätzliche Kosten, da der neue Wert zur zuständigen Zone geschickt werden muss.

Die Autoren geben den durchschnittlichen Aufwand zum Einfügen eines Datums in einem dichten Sensornetz mit  $N$  Knoten als proportional zu  $\sqrt{N}$  an, da im Wesentlichen geographisches Routing (GPSR) benutzt wird, um eine Nachricht zuzustellen und diese im Durchschnitt  $\sqrt{N}$  Knoten bis zum Ziel durchläuft. Zur Aufwandsabschätzung wird nur die Anzahl der verschickten Nachrichten herangezogen, da die Kosten für lokale Berechnungen im Vergleich mit den Kommunikationskosten zu vernachlässigen sind.

Desweiteren wird der durchschnittliche Aufwand der Zustellung einer Anfrage betrachtet. Die Anzahl der Nachrichten, die zur Zustellung der Antwort benötigt werden, wird für alle Verfahren als gleich angenommen und daher nicht weiter berücksichtigt. Die Zustellungskosten hängen bei DIMs offensichtlich von der Größe der abgefragten Wertebereiche ab.

Falls alle Anfrageintervalle gleich wahrscheinlich sind, also beispielsweise eine Punktanfrage die gleiche Wahrscheinlichkeit besitzt wie eine Anfrage, die den gesamten Wertebereich umfasst, so ergibt sich im Vergleich mit dem Fluten der Anfrage keine Verbesserung, denn der asymptotische Aufwand ist mit  $\mathcal{O}(N)$  identisch.

Für andere Wahrscheinlichkeitsverteilungen<sup>1</sup> ergibt sich jedoch ein Effizienzgewinn zu  $\mathcal{O}(\sqrt{N})$ . In diesem Fall sind die Kosten für die Anfragebearbeitung also in etwa gleich zu denen, die bei der Verarbeitung einer Aktualisierung entstehen.

Offensichtlich ist es nicht sinnvoll, DIMs zu verwenden, wenn Sensoren laufend neue Werte liefern, aber Anfragen relativ selten sind. Wann ist der Einsatz von DIMs also vorteilhaft? Den Autoren zufolge „ist es einfach zu zeigen, dass DIMs die Performanz des Flutens dann übertreffen, wenn das Verhältnis zwischen der Anzahl der Einfügeoperationen und der Anzahl der Anfragen kleiner als  $\sqrt{N}$  ist“. Auf  $\sqrt{N}$  neue Werte muss also mindestens eine Anfrage folgen.

In einer recht umfangreichen Simulation werden DIMs mit GHTs [5] verglichen. Obwohl beide Verfahren einen ähnlichen asymptotischen Aufwand für Einfügeoperationen und die Zustellung von Anfragen (beides  $\mathcal{O}(\sqrt{N})$ ) besitzen, schneiden DIMs in beiden Kategorien deutlich besser ab. Dies liegt vor allem daran, dass GHTs nur einen (z. B. den ersten) Attributwert bei der Abbildung eines Datums auf einen Sensorknoten berücksichtigen. Dadurch hängt die Anzahl der Teilanfragen ausschließlich von dem angefragten Bereich dieses Attributs ab.

Neben der Simulation wurde das Verfahren auch auf einer Linux-basierten Plattform für PDAs und PC/104-Rechner implementiert. Die Implementierung diente allerdings eher als Proof-of-Concept als zur ernsthaften Evaluation. Als nächsten Schritt sehen die Autoren eine Portierung auf MICA Motes und eine Integration in ein Datenbanksystem für Sensornetze wie TinyDB [7] oder Cougar [8].

---

<sup>1</sup> Gleichverteilung mit begrenzter Intervallgröße, Algebraische Verteilung, Exponentialverteilung

## 4 Stufenförmig verteilte Informationen in Sensornetzen

Das Prinzip der „*Fractionally Cascaded Information in a Sensor Network*“ [2] ist es, Informationen so über das Sensornetz zu verteilen, dass jeder Sensor einen Teil der Daten entfernter Sensoren kennt. Dabei soll räumliche Lokalität gegeben sein, d. h. ein Sensor hat mehr Information über nahe Sensoren als über weiter entfernte.

Als Ziel wird wiederum die effiziente Bearbeitung von Bereichsanfragen (vgl. Abschnitt 2) angegeben. Die Indexstruktur soll vor allem kontinuierliche Anfragen mit Lokalitätsinformationen beschleunigen. Dies sind Anfragen, die fortlaufend nach Ablauf eines vorgegebenen Zeitintervalls ausgewertet werden sollen und zusätzlich eine Einschränkung bezüglich des geographischen Bereichs vorsehen.

### 4.1 Verteilung der Daten auf die Knoten

Als zugrundeliegende Datenstruktur wird ein Quadtree [9] verwendet. Dadurch wird das als quadratisch angenommene Sensornetz rekursiv jeweils in 4 gleichgroße Quadranten eingeteilt, bis sich in jedem Quadrat nur noch maximal ein Sensor befindet (siehe Abb. 7). Die Sensoren bilden die Blätter des Baumes.

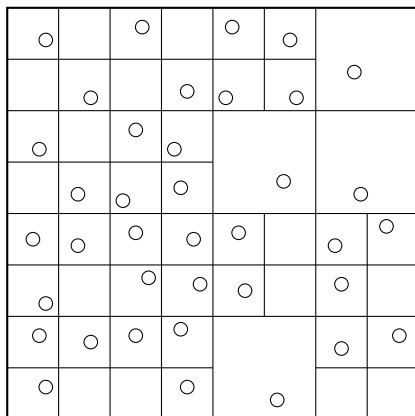


Abb. 7. Blätter des Quadtrees

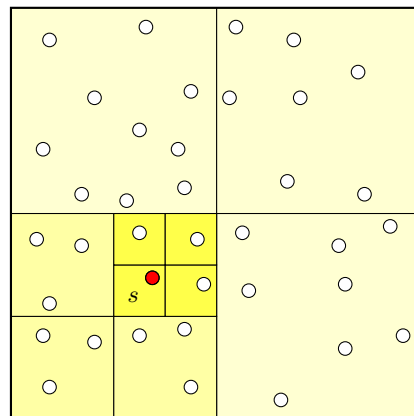


Abb. 8. Sicht des Sensors  $s$

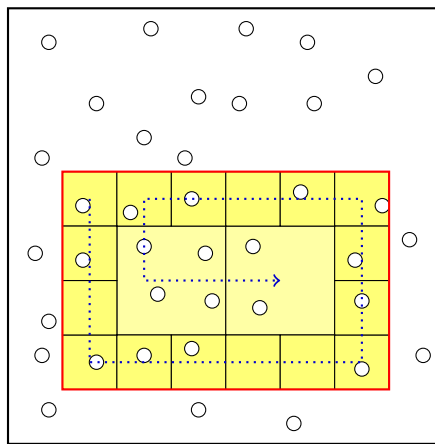
Der allgemeine Aufbau ist nun wie folgt: In jedem Knoten des Quadtrees werden über alle Sensoren in seinem Bereich aggregierte Daten gespeichert. Jeder Sensorknoten  $s$  – entspricht einem Blatt im Quadtree – speichert nun die Daten aller Geschwister eines Knoten auf dem Pfad von  $s$  zur Wurzel.

Wie man im Abb. 8 sieht, impliziert dies eine Aufteilung in Kacheln (*tiles*). Der Sensor  $s$  speichert die Daten seiner Geschwisterknoten (kleinste Quadrate), der Geschwister seines Vaterknotens (nächstgrößere Quadrate), usw. bis zur Wurzel.

*Beispiel 9.* Eine konkrete Instanz dieses allgemeinen Schemas stellt ein Sensornetz dar, in dem die Sensorknoten die Temperatur messen und als Aggregationsfunktion das Minimum bzw. Maximum verwendet wird. Dann speichert jeder Sensor  $s$  die minimale ( $\min_t(u)$ ) und maximale Temperatur ( $\max_t(u)$ ) der jeweiligen Kachel  $u$ . Eine Anfrage besteht in diesem Fall aus drei Komponenten  $(q, R, T)$ : dem Sensor  $q$ , der die Anfrage absetzt, dem rechteckigen Zielbereich  $R$  und dem Temperaturbereich  $T = [T_1, T_2]$ . Typische Anfragen lauten dann etwa „Wie viele Sensoren mit einer Temperatur zwischen  $20^\circ\text{C}$  und  $30^\circ\text{C}$  befinden sich im Bereich  $R$ ?“  $(q, R, [20, 30])$  oder „Liste alle heißen Sensoren im Bereich  $R$ .“  $(q, R, [T, \infty))$ .

#### 4.2 Ablauf einer Anfrage

Zunächst benötigen wir eine Definition: Ein Knoten  $u$  des Quadrees heißt *kanonisch*, falls der Bereich von  $u$  komplett in  $R$  enthalten ist, der Bereich des Vaterknotens  $p(u)$  jedoch nicht. In Abb. 9 sind die kanonischen Knoten für den rot markierten Zielbereich angegeben.



**Abb. 9.** Kanonische Knoten des Quadrees

Zur Bearbeitung einer Anfrage wird diese zunächst von der Quelle zur angefragten Region gesandt. Dort wird die Menge der kanonischen Knoten  $U$  bestimmt und für jeden Knoten  $u \in U$  ein beliebiger Sensor in seinem Bereich besucht. Die Autoren machen keine Aussage darüber, wie sich die Besuchsreihenfolge auf die Kommunikationskosten auswirkt. Sie wird als beliebig angenommen und kann wie in Abb. 9 spiralförmig sein. Für jeden kanonischen Knoten wird nun, falls notwendig, sein Unterbaum abgelaufen und passende Sensoren gemeldet.

Angenommen, man interessiert sich im Temperaturbeispiel (vgl. Beispiel 9) für heiße Sensoren in einem gewissen Bereich  $R$ , also  $(q, R, [T, \infty))$ . Die Anfrage

durchläuft nun für jeden kanonischen Knoten  $u$  einen Sensor  $s$ . Falls  $\max_t(u) < T$  trägt der Knoten nichts zur Lösung bei und kann übersprungen werden. Ansonsten werden rekursiv die Kinder von  $u$  besucht. Das Ergebnis lässt sich in diesem Fall aus den Teilergebnissen der kanonischen Knoten zusammensetzen.

### 4.3 Bewertung des Verfahrens

Das Verfahren benutzt eine Quadtree-ähnliche Struktur, in der für jeden Knoten vorberechnete Aggregatwerte bestimmter Bereiche des Sensornetzes vorgehalten werden. Diese sind nach dem Prinzip des *Fractional Cascading* verteilt: Das Wissen eines Sensorknotens über entfernte Bereiche des Netzes nimmt mit der Entfernung ab. Dazu speichert jeder Sensor Informationen im Umfang proportional zur Höhe des Quadtrees, also  $\mathcal{O}(\log n)$ , ab. Durch die Replizierung der Daten eines Knotens auf alle Sensoren in seinem Bereich wird vermieden, dass einzelne Knoten zu Kommunikationsengpässen werden.

Beschleunigt werden Anfragen, bei denen die gespeicherten Aggregatwerte genutzt werden können, um frühzeitig irrelevante Sensoren auszuschließen. Dies ist in Beispiel 9 für Anfragen der Art  $[T, \infty)$  bzw.  $(-\infty, T]$  der Fall. Für allgemeine Bereichsanfragen der Form  $[T_1, T_2]$  ist dies unter Umständen jedoch nicht so. Wenn nur der maximale/minimale Wert eines Quadtreenotens  $u$  vorberechnet und gespeichert wird, ist es möglich, dass sowohl das Maximum größer ist als die Obergrenze des gesuchten Intervalls als auch das Minimum kleiner als die zugehörige Untergrenze. In einem derartigen Fall müssen also die Kinder von  $u$  besucht werden, möglicherweise fallen aber keine ihrer Werte in das Anfrageintervall.

Den Autoren zufolge ist dies in einem kontinuierlichen Feld wie der Temperatur, in dem sich Messwerte nicht abrupt ändern, jedoch selten der Fall. Um solche Anfragen in jedem Fall zu beschleunigen, müsste jeder Quadtreenotens anstelle von Maximum und Minimum ein Histogramm der Werte speichern, was jedoch ein Vielfaches an Speicherbedarf erfordert.

Einen weiteren Kritikpunkt sehe ich bei der Verarbeitung von Updates. Wenn sich der Wert eines Sensors ändert, muss dies im Quadtree nach oben propagiert werden. Falls die Knoten wie im Beispiel das Maximum bzw. Minimum speichern, kann die Aktualisierung stoppen, sobald sich der Wert nicht mehr ändert. Im schlimmsten Fall setzt sich die Aktualisierung jedoch bis zur Wurzel fort. Da jeder Sensor den Wert des Wurzelknotens speichert, müssen dann alle Sensoren benachrichtigt werden.

Das Beispiel zeigt gleichzeitig alle Vorteile des Verfahrens auf. Messen die Sensoren einen Wertebereich ähnlich der Temperatur und ist man an „heißen“ bzw. „kalten“ Sensoren in einem gewissen Bereich interessiert, lässt sich durch die Verwendung des *Fractionally Cascading* ein deutlicher Effizienzgewinn erzielen. Ist man an den konkreten Messwerten der Sensoren interessiert, könnte sich die Verwendung und Instandhaltung des Index eher nachteilig auswirken.

## 5 Sensorbäume

Die Idee hinter Sensorbäumen (*sensor trees*) [3] besteht darin, die Sensoren in einem verteilten R-Baum [10] zu organisieren, um die Bearbeitung von Aggregationsanfragen zu beschleunigen. Die Sensoren bilden die Blätter des Baumes (siehe Abb. 10), die inneren Knoten speichern zusätzlich über ihre Kinder vor-aggregierte Werte.

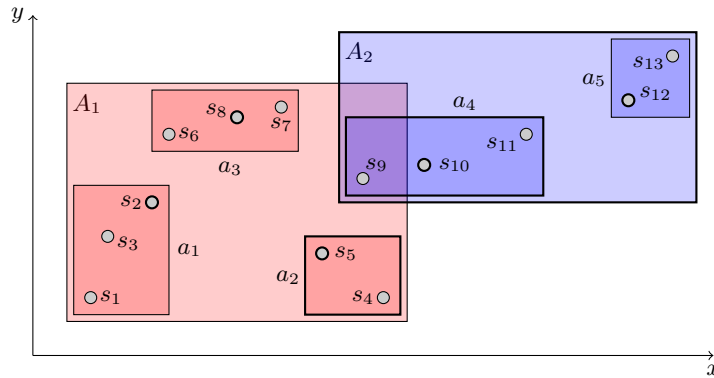
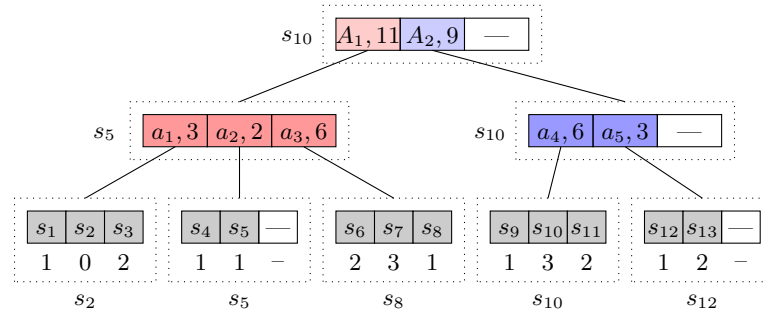


Abb. 10. Aufbau eines Sensorbaums für ein SUM-Aggregat

### 5.1 Aufbau des Baumes

Ein R-Baum teilt einen  $n$ -dimensionalen Raum in hierarchisch geschachtelte, potentiell überlappende Rechtecke (siehe Abb. 10). Der so entstehende höhen-balancierte Baum basiert auf einem B-Baum (bzw. B\*-Baum). Die Blätter enthalten Verweise auf die zu speichernden Daten, in diesem Fall die Sensoren. Jeder Knoten des R-Baums entspricht einem  $n$ -dimensionalen Rechteck und wird auch als *Bounding Box* oder *Cluster* bezeichnet.

Wie in einem B-Baum haben die Knoten eine bestimmte Kapazität, d. h. sie müssen mindestens  $n$  und dürfen höchstens  $N$  Einträge enthalten. Für jeden Eintrag  $v$  in einem inneren Knoten  $u$  ist das durch  $u$  repräsentierte Rechteck das kleinste Rechteck, welches das durch  $v$  repräsentierte Rechteck enthält. Zusammenfassend ausgedrückt, enthält jeder Knoten des Baumes einen Eintrag pro enthaltenem Rechteck zusammen mit dem zugehörigen Aggregatwert.

Zunächst werden ein paar Annahmen bezüglich des Sensornetzes gemacht. Jeder Sensorknoten muss eine eindeutige ID besitzen. Das Sensornetz muss verbunden sein, die Kosten für die Kommunikation zweier Knoten mit Abstand  $d$  werden als  $\mathcal{O}(d)$  angenommen.

Benötigt wird nun eine geeignete Repräsentation der inneren Knoten des Sensorbaums durch die tatsächlich vorhandenen Sensorknoten. Aus Effizienzgründen sollte diese Auswahl von der Routingstruktur abhängig gemacht werden, da sonst zusätzliche Kosten für die Kommunikation entstehen. Die Autoren schlagen jedoch lediglich ein relativ abstraktes Verfahren vor, das keinen Bezug auf ein konkretes Routingverfahren nimmt.

Jeder Sensorknoten  $v$  speichert sein Level  $l.v$  im Baum, das der höchsten Ebene des Baums entspricht, an der  $v$  beteiligt ist. Das Level jedes Sensors ist mindestens 0, da er zumindest an der untersten Ebene des Baums teilnimmt. Knoten mit Level  $i$  kooperieren zur Konstruktion der nächsthöheren Ebene  $i+1$ . Für jedes Level  $i$  speichert ein Sensorknoten  $v$  außerdem einen Verweis  $p.v(i)$  auf seinen Vaterknoten und eine Menge von Verweisen  $c.v(i)$  auf seine Kinder.

Der Sensorbaum wird in einem *Bottom-up*-Ansatz aufgebaut, d. h. der Aufbau beginnt auf Ebene 0 des Baumes. Jeder Sensorknoten mit Level  $i$ , der noch keinen Verweis auf einen Vaterknoten  $p.v(i)$  hat, versucht hierzu einen Nachbarn mit Level  $i+1$  zu finden.

Dazu sucht er mit zunehmend größerem Radius in seiner Umgebung. Findet er einen solchen Knoten  $u$  mit Level  $i+1$ , setzt er  $p.v(i) = u$  und benachrichtigt  $u$ , damit dieser einen Verweis auf  $v$  in  $c.u(i)$  ergänzen kann. Falls kein solcher Knoten  $u$  existiert,  $v$  jedoch  $n$  vaterlose Knoten auf Level  $i$  gefunden hat, wird  $v$  selbst zum Vater dieser Knoten und erhöht sein Level auf  $i+1$ . Dieser Algorithmus wird nun solange fortgesetzt, bis der Sensorbaum vollständig aufgebaut ist.

Da der Algorithmus gleichzeitig ohne Synchronisation auf den Knoten abläuft, kann es vorkommen, dass ein innerer Knoten zwischenzeitlich mehr als  $N$  Kinder hat. In einem solchen Fall wird das Cluster mit Hilfe der *split*-Operation des R-Baums in zwei neue Cluster aufgeteilt.

Außerdem ist es möglich, dass sich zwei oder mehr Knoten gleichzeitig zum Vater eines Clusters erklären. In diesem Fall haben beide Knoten Level  $i+1$  und jeweils einen Verweis auf den anderen Knoten in  $c.v(i)$ . Dies wird dadurch aufgelöst, dass der Knoten  $v$ , der zuerst von einem anderen Knoten  $u$  mit  $l.u = i+1$  kontaktiert wird, sein Level um 1 verringert und dem Cluster von  $u$  beiträgt.

*Beispiel 10.* In Abb. 10 wird ein Sensorbaum veranschaulicht, der die Summe als Aggregationsfunktion benutzt. Die Repräsentanten der inneren Knoten sind durch die Angabe des zuständigen Sensors neben dem gestrichelten Kasten, der

einen Knoten umfasst, gekennzeichnet. Beispielsweise wird der Wurzelknoten durch den Sensor  $s_{10}$  vertreten, der außerdem  $A_2$  und  $a_4$  verwaltet. Neben den Verweisen auf die Kinder wird in den Einträgen der inneren Knoten die Summe der Messwerte der Kindknoten gespeichert.

## 5.2 Bearbeitung einer Anfrage

Jeder Knoten besitzt einen Verweis auf seinen Vaterknoten und Verweise auf seine Kinder. Der Baum ist somit doppelt verkettet, insbesondere kann jeder Sensorknoten eine Anfrage absetzen.

Anfragen, die sich auf den vorberechneten Aggregatwert beziehen, werden entlang der Baumhierarchie zu demjenigen Sensorknoten weitergeleitet, der die Wurzel des Sensorbaumes repräsentiert. Dort wird die Anfrage dann wie bei einem R-Baum bearbeitet.

*Beispiel 11.* In Beispiel 10 speichern die inneren Knoten die Summe als voraggregierten Wert. Angenommen, Sensorknoten  $s_1$  interessiert sich für die Summe der Messwerte im Bereich  $R$  aus Abb. 11.

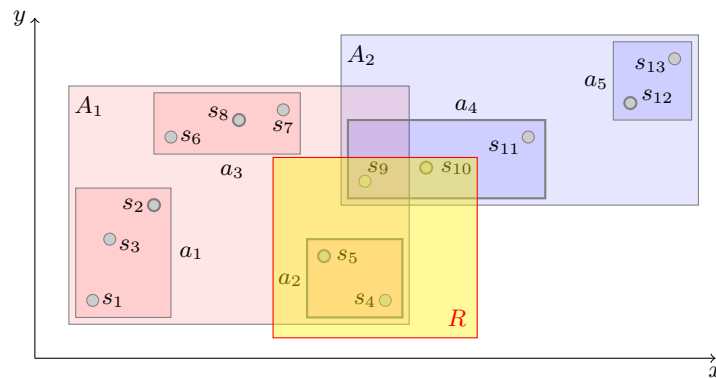


Abb. 11. Zielbereich einer Anfrage

Zur Bearbeitung der Anfrage meldet sich  $s_1$  zunächst bei seinem Vaterknoten  $a_1$ , der durch  $s_2$  vertreten wird.  $s_2$  stellt fest, dass  $R$  sich nicht mit  $a_1$  überschneidet und sendet die Anfrage daher an  $s_5$  ( $A_1$ ) weiter. Der Sensorknoten  $s_5$  erkennt nun, dass  $a_2$  im gesuchten Bereich liegt und er selbst zuständig ist.  $s_5$  kann also bereits ein Teilergebnis an  $s_1$  senden.

Da in R-Bäumen jedoch prinzipiell – auch hier im Beispiel – Überlappungen der Rechtecke möglich sind, wird  $s_5$  die Anfrage in jedem Fall an die Wurzel ( $s_{10}$ ) weiterleiten. Dort wird festgestellt, dass neben  $A_1$  auch  $A_2$  eine Überschneidung mit dem Zielbereich aufweist. Da  $s_{10}$  selbst sowohl  $A_2$  als auch  $a_4$  verwaltet, kann er nun die Anfrage direkt bearbeiten und sein Ergebnis an  $s_1$  senden.

### 5.3 Bewertung des Verfahrens

Sensorbäume organisieren ein Sensornetz in einer Struktur, die eng mit R-Bäumen verwandt ist. Dadurch ist es möglich, Anfragen, die sich auf bestimmte Bereiche des Netzes beziehen, gezielt den Knoten zuzustellen, deren Bounding Box eine Überlappung mit dem Anfragebereich aufweist. Dies ist ein deutlicher Vorteil im Vergleich zum Fluten der Anfrage an alle Sensoren.

Zusätzlich werden voraggregierte Werte in den inneren Knoten gespeichert. Diese können unter bestimmten Umständen die Bearbeitung von Anfragen beschleunigen. Im Beispiel aus Abb. 10 wird die Summe als Aggregat verwendet. Es ist aber jede andere holistische Aggregationsfunktion denkbar.

Wie jede hierarchische Struktur auf Sensornetzen haben auch Sensorbäume das Problem, dass Sensoren, die Knoten auf hohen Ebenen der Hierarchie repräsentieren, zu Kommunikationsengpässen werden. Sie werden in die Bearbeitung von weitaus mehr Anfragen einbezogen als Sensoren, die sich in der Hierarchie weiter unten befinden. Dies führt zu einem früheren Ausfall der betroffenen Knoten.

Die Autoren geben leider keine konkreten Zahlen zum Effizienzgewinn an, auch eine Simulation oder Implementierung wurde nicht vorgenommen.

## 6 Fazit

In dieser Seminararbeit wurden drei Ansätze für Indizes auf Sensornetzen vorgestellt. Alle drei Datenstrukturen werden im Netzwerk aufgebaut und verwaltet. Anfragen können von jedem Sensorknoten abgesetzt werden. Trotzdem ergeben sich beträchtliche Unterschiede. Während DIMs den Wertebereich in Zonen aufteilen, benutzen Sensorbäume und Fractionally Cascading eine hierarchische Struktur und vorberechnete Aggregatwerte.

Es existieren zahlreiche weitere Lösungen für verteilte Indexstrukturen auf Sensornetzen. Wie schon bei der genaueren Untersuchung der vorgestellten Datenstrukturen deutlich wurde, unterscheiden sich die Verfahren häufig je nachdem, welches Problem damit gelöst werden soll. Interessiert man sich nur für Sensoren, deren Messwerte eine bestimmte Schwelle überschreiten? Im gesamten Netzwerk oder einem bestimmten Bereich? Will man die Korrelation der Daten mehrerer Dimensionen ausnutzen?

Deutlich wurde also vor allem, dass keine Patentlösung existiert, die auf alle Szenarien anwendbar ist. Die Eignung eines Verfahrens hängt in hohem Maße von der Zielsetzung und der gewünschten Einsatzumgebung ab. In der Praxis ist eine Kombination von mehreren verschiedenen Indexstrukturen denkbar, von denen jede gezielt für einen Zweck eingesetzt wird.

## Literatur

1. Li, X., Kim, Y.J., Govindan, R., Hong, W.: Multi-dimensional range queries in sensor networks. In: SenSys '03: Proceedings of the 1st international conference

- on Embedded networked sensor systems, New York, NY, USA, ACM Press (2003) 63–75
2. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L.: Fractionally cascaded information in a sensor network. In: IPSN '04: Proceedings of the third international symposium on Information processing in sensor networks, New York, NY, USA, ACM Press (2004) 311–319
  3. Park, S.Y., Bae, H.Y.: A distributed spatial index for time-efficient aggregation query processing in sensor networks. In: ICCS '05: Proceedings of the 5th International Conference on Computational Science. Volume 3514 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer (2005) 405–410
  4. Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J., Culler, D.: An analysis of a large scale habitat monitoring application. In: SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, New York, NY, USA, ACM Press (2004) 214–226
  5. Ratnasamy, S., Karp, B., Yin, L., Yu, F., Estrin, D., Govindan, R., Shenker, S.: Ght: a geographic hash table for data-centric storage. In: WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, New York, NY, USA, ACM Press (2002) 78–87
  6. Karp, B., Kung, H.T.: GPSR: greedy perimeter stateless routing for wireless networks. In: MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking, New York, NY, USA, ACM Press (2000) 243–254
  7. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The design of an acquisitional query processor for sensor networks. In: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, New York, NY, USA, ACM Press (2003) 491–502
  8. Yao, Y., Gehrke, J.: The cougar approach to in-network query processing in sensor networks. SIGMOD Rec. **31** (2002) 9–18
  9. Finkel, R.A., Bentley, J.L.: Quad trees: A data structure for retrieval on composite keys. Acta Informatica **4** (1974) 1–9
  10. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: SIGMOD '84: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, ACM Press (1984) 47–57