Efficient Support for Multi-Resolution Queries in Global Sensor Networks

Andreas Benzing Boris Koldehofe Kurt Rothermel

Distributed Systems Group, Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart, Germany
{firstname.lastname}@ipvs.uni-stuttgart.de

ABSTRACT

Stream processing has evolved as a paradigm for efficiently sharing and integrating a massive amount of data into applications. However, the integration of globally dispersed sensor data imposes challenges in the effective utilization of the IT infrastructure that forms the *global sensor network*. Especially simulations require the integration of sensor streams at widely differing spatial and temporal resolutions. For current stream processing solutions it is necessary to generate a separate data stream for each requested resolution. Therefore, these systems suffer from high redundancy in data streams, wasting a significant amount of bandwidth and limiting their scalability.

This paper presents a new approach to scalable distributed stream processing of data which stems from globally dispersed sensor networks. The approach supports applications in establishing continuous queries for sensor data at different resolutions and ensures efficient bandwidth usage of the data distribution network. Unlike existing work in the context of video stream processing which provides multiple resolutions by establishing separate channels for each resolution, this paper presents a stream processing system that can efficiently split/combine data streams in order to decrease/increase their resolution without loss in data precision. In addition the system provides mechanisms for load balancing of sensor data streams that allow efficient utilization of the bandwidth of the global sensor network.

Keywords

DSPS, Global Sensor Network, Indexing, Query Processing

1. INTRODUCTION

Today, sensors are deployed on a global scale in virtually every area. Examples are weather stations, satellites, and sensors on airplanes that together capture the state of the atmosphere. In order to support scalable access to this huge amount of information by possibly many users and applications, the information is often provided in an aggregated form as human readable information. While this is a valid choice for information purposes, further processing, for example in simulation applications, requires a set of data points provided at regular intervals without gaps. The requirements for the resolution at which data points are provided can differ significantly between multiple applications. For instance, a simulation might monitor a large area at coarse resolution to detect emerging hurricanes. Once a hurricane is detected, a higher resolution of data is required to monitor the movement and the wind field of the hurricane for the area in which the hurricane was detected.

Clearly, a stream processing solution should ensure that data streams are distributed according to the respective requested resolution. Otherwise, a significant amount of bandwidth would be wasted by unnecessary transmissions of data points on the path to the clients. However, once multiple applications, like hurricane and wind field monitoring, require the processing of overlapping sensor streams at different resolutions, stream processing systems also need to avoid redundant transmissions of shared sensor data.

While most recent stream processing solutions (cf. [4, 5]) provide efficient solutions in sharing sensor streams with respect to the same resolution, there is only limited support in providing data streams at multiple resolutions. Other systems, especially for video streaming, would simply require to generate separate data streams for each resolution.

In this paper we propose and evaluate a system for the efficient distribution of sensor data, especially in the context of real-time simulation applications that rely on sensor data input. Queries to the system are indexed using the GBD-Tree [17] to allow for efficient lookup and reuse of already established data streams. The indexing is extended to provide different resolutions to users and eliminate redundant data streams. The extended indexing schema furthermore allows to split and balance the network load induced by queries and this way achieve a highly efficient load balancing. Our evaluations show that with the adaptation to multi-resolution operation, the bandwidth required for load balancing can be reduced by over 40%.

The remainder of the paper is structured as follows: first, background information and related work to the presented

COMSWARE 2011, July 04-07, Verona, Italy

DOI 10.4108/comsware.2011.11

approach is given in Section 2. Then, Section 3 describes the underlying system and query model. In Section 4 the query processing and overlay management is presented in detail. Evaluations are shown in Section 5 and Section 6 concludes the paper.

2. RELATED WORK

To exploit the advantages of in-network processing of data, several distributed stream processing systems (DSPS) like Borealis [1] have been proposed. However, its query interface requires manual stream modeling and distribution of operators to available nodes. Other approaches like IrisNet [15] and HiFi [12] focus on the special properties of sensor data and provide data filtering and preprocessing close to the sensors. Virtual sensors were introduced in the GSN middleware [2] to allow easy provision of preprocessed sensor data. Hourglass [22] provides robust so called circuits on intermittent connections between nodes.

Although IrisNet, HiFi, GSN, and Hourglass all support information source lookup and automatic routing of data streams, they lack the reuse of streams and intuitive range querying as required for global sensor data. An approach to integrate the routing of data streams with an intuitive query mechanism was presented in [4]. The reduction of data rates using a network of adaptive filters has been addressed in [18]. However, the efficient provision of multiple resolutions at the same time still cannot be provided.

SBON [19] introduces a layer between the DSPS and the physical network to optimize placement for network usage. The approach was improved towards optimal mapping of operators with respect to the underlying network [21]. Although operators might be placed on the same node, explicit reuse of data streams is not supported by this approach.

Contrary to these DSPS, our approach is to exploit the similarities in data streams by actively avoiding redundant data transmission. Publish/subscribe [6, 7, 11] has emerged as a generic powerful many-to-many communication paradigm. It is applied for efficient decoupling of data sources and subscribers to data streams mostly in event processing systems. More recent approaches also consider the delay and bandwidth constraints of subscribers [23, 24]. All these approaches, however, accept a certain amount of false positives to provide scalable management of highly dynamic subscriptions. Unlike publish/subscribe, this new approach is able to split and reunite data streams. This way, it can provide better matching with very few false positives which is required for the high bandwidth used by global sensing systems.

To handle huge amounts of data, especially in the context of climate simulation, systems have been proposed to split gridded data into multiple smaller chunks. DataCutter [5] provides a network of filters to allow distributed filtering and processing of data. However, it does not allow combination of streams. Therefore, new data streams that are not a subset of existing streams have to be generated from the source and introduce additional redundancy in data transmission. Another system for climate simulation data is the Earth System Grid-I [10]. It provides data access to archived data rather than real-time data streams and is therefore not suited for a globally distributed sensing system.

Scribe [9] and SplitStream [8] provides application layer multicast routing in peer-to-peer systems. Both are related to this work as data is distributed from single sources to many destinations. However, Scribe does not address the problem of providing data at multiple resolutions to the clients. For SplitStream, a suitable encoding is assumed that can be divided to split the actual data stream into smaller chunks. While this approach is well suited for the dissemination of video streams where certain channels can be addressed, it lacks support for range queries on gridded data.

3. SYSTEM AND QUERY MODEL

We consider stream processing in the context of a distributed broker infrastructure. This infrastructure gathers and preprocesses sensor information to provide a complete grid of data points from all over the world. Applications can pose continuous queries to the system to retrieve real-time data streams from a specified location with a certain resolution.

For example, a weather forecast application might require detailed data for a small region of interest. In addition, it requires data of a large area to account for the global state of the weather. Due to limitations in computational power, only a reduced resolution of the data available is used for the estimation of the future weather state. This reduced resolution for global data is usually one order of magnitude coarser than the high resolution used for local data which covers data points at a distance of a few kilometers (e.g. [14]). Another example might be a particle simulation which provides information about harmful substances in an urban region in real-time. In contrast to the weather forecast, such a simulation requires data with high resolution for a smaller constrained region. A mixture of both scenarios has to be considered for a flexible and versatile system as described in the following sections.

Specific components of the system and their properties are presented in Section 3.1. A detailed description of the underlying query model is given in Section 3.2. Based on the properties of the system components and the query model, a problem statement is then provided in Section 3.3.

3.1 System Components

The Global Sensor Grid system described in this paper is formed by a set of *brokers* as depicted in the upper part of Figure 1. Each broker is responsible for collecting sensor data of a certain type of sensor from a dedicated region. However, one broker can be responsible for multiple sensor types and regions. In other words, for each type of sensor the entire world is divided into smaller spatial regions. Each of the smaller regions is independently assigned to a certain broker.

Using this sensor information and the information from the boundary of the neighboring regions, each broker computes a grid of data points for its region. The brokers cooperate to deliver the sensor data to the clients by replicating data of highly loaded regions. Requests, which can be posed to arbitrary brokers, can then be distributed to multiple brokers, avoiding the overload of a single broker.

The partition of the global area into regions and the mapping of regions to the brokers provides a basic overlay topology.

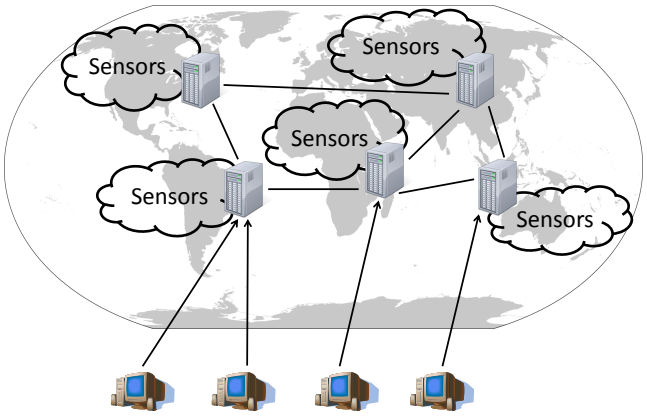

Figure 1: The network of brokers, each broker gathers the sensor information for its assigned sensor types and region. The clients in the lower part of the picture pose their queries to any broker.

It consists of the connections from one broker to the brokers managing the neighboring regions. This topology is used for query routing and lookup of the broker which provides the data for a certain query. Although this overlay is not optimized to resemble the underlying network topology, brokers typically reside topologically close to the sensor data sources in the network.

3.2 Query Model

A query q is a tuple composed by its region R_q and the set of attributes A_q of interest:

$$q = (R_q, A_q)$$

The query is processed by the broker network and then answered with a data stream containing all data points covered by the query. A data point d is thereby specified by a 2D location information x_d and y_d, an attribute id aid_d and the value of that attribute val_d at a certain point in time t_d. The internal representation of queries for further processing is described in Section 4.1.

Besides geographical bounds of the area of interest, the query region also contains information about the requested resolution. The query region of query q is represented by a tuple

$$R_q = (x_{min}, x_{max}, y_{min}, y_{max}, res_{xy}, res_t).$$

A 2D coordinate representation is used to describe the physical space and the queried regions. x_{min} and x_{max} describe the lower and upper bound for the latitude of the queried area. y_{min} and y_{max} limit the longitude correspondingly. The last two values res_{xy} and res_t give the spatial and temporal resolution of the query as a fraction of the maximum available data. For example, if $res_t = 0.25$ only every fourth data update is sent to the client node.

Clearly, a query can be decomposed into multiple disjoint subqueries of smaller size. The union of the respective subregions forms the entire area of the original query. Pairwise intersections of subregions, however, are completely empty. This property will later be used to divide queries into smaller parts where certain subqueries are common to multiple clients. The proposed system efficiently distributes the load of such common subregions to avoid bottlenecks.

In addition to location and resolution information, a client can also specify a set of attributes which should be delivered by the system. More formally, A_q specifies the set of attribute IDs which the user is interested in. The presented system also allows the user to specify a lower and upper bound on each of the attributes. By restricting the value of the attribute to a certain interval, similar to publish/subscribe, users can further constrain the volume of the data stream generated. However, this approach focuses on data distribution to applications which usually require the full available spectrum for an attribute of interest. Therefore, throughout the rest of the paper, we do not consider additional filters for a specific attribute.

3.3 Problem Statement

The goal of the presented system is to provide as many clients as possible with sensor data despite highly imbalanced query regions. Since each broker has only a limited amount of bandwidth available for data distribution, the load has to be distributed among brokers. The basic function of the presented system is therefore to establish replication data streams between brokers when necessary with possibly low overhead.

This function is considered optimal if the total amount of data delivered to client nodes is maximized without exceeding the bandwidth locally available at every broker. Formally, let B be the set of brokers and let Q_i be the set of queries to answer by broker i. The set $Q = \bigcup_i Q_i$ contains all queries of the whole system. $|q|$ denotes the size of a query $q \in Q$, i.e. the bandwidth required to serve it. Additionally, let C_i be the set of outgoing connections for replication from broker i where $|c|$ denotes the size of a connection analogously to $|q|$. Finally, let a_i be the total bandwidth available at broker i. An optimal solution then provides the maximum total query size without exceeding the local bandwidth at each broker. This can be described as follows:

$$\textbf{maximize} \quad \sum_{q \in Q} |q|$$

$$\textbf{subject to} \quad \forall b \in B : \sum_{c \in C_b} |c| + \sum_{r \in Q_b} |r| \leq a_b$$

The next section presents our approach to data stream management based on the given system model and optimization goal.

4. DATA STREAM MANAGEMENT

Our approach for data stream management consists of two major parts: query processing and load distribution. Query processing starts by indexing the queries using an extended version of the GBD-Tree [17]. A query is split in several parts which refer to data of distinct geographical areas. For each part of a query the index determines the brokers which

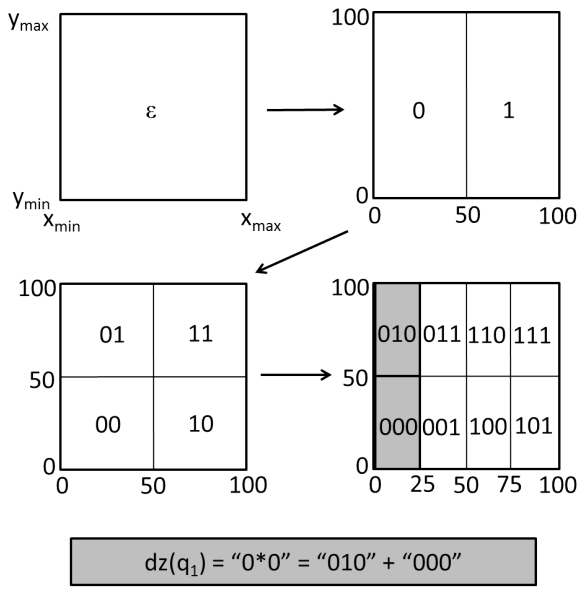

Figure 2: Example for generation of DZ expressions using spatial indexing.

can provide the requested data. The index is maintained in a completely decentralized fashion. By contacting any broker in the network it is possible to determine just from a set of neighbors maintained at each broker how to direct the query subsequently to the broker responsible for the data streams. In order to support streams at multi-resolution we have extended the index in a way that also covers the resolution information in addition to the location specified in the queries. Using this detailed information, the system can adapt to high demand not only for specific regions but also with the required resolution.

Since brokers have only limited bandwidth available, overload situation can occur at a broker. In the presence of an overload situation the parts of the data streams which are of high interest to many users are replicated to other appropriate brokers. Replicated brokers will help to respond to requests and contribute to balance the load. The number of replicas is thereby adapted according to the current load situation.

In the following we present the approach in more detail. Section 4.1 gives an overview how the basic index structure works. In Section 4.2 we present how this index is extended to support multi-resolution data streams. Finally, Section 4.4 shows how our approach balances the load on the data streams and selects appropriate brokers for replication.

4.1 Query Indexing

To efficiently identify, for each part of a query, the brokers responsible for streaming the requested data, we use a structured spatial index [13], specifically the GBD-Tree [17] (Generalized BD-Tree). The GBD-Tree was originally proposed to index very large spatial databases with high dimensional data. This index allows fast and efficient detection of areas which are of relevance to multiple queries. Hereby each part of a query is associated with a node in the GBD-Tree and the respective subtree can be used to extract the information of interest.

The GBD-tree ensures that the area associated with child nodes is strictly contained in the area of any given parent. Hence, it supports quick identification of reuse relationships for the corresponding dissemination structures. Furthermore, the containment relationship releases the system from dealing with the problems of managing overlapping queries. Such overlap occurs when using an index like the R*-Tree [3] which allows arbitrary intersections of MBRs. To track intersections in an R*-Tree, all potential subtrees have to be completely traversed which induces a high overhead. In addition, the large number of possible intersections leads to ambiguity in the creation of dissemination structures.

Nodes in the GBD-tree are represented by DZ expressions. These are generated in a recursive process as depicted in Figure 2. DZ expressions are formed by a string of characters 0, 1, and *. An empty DZ expression (ϵ) represents the whole space managed by the system. ϵ covers all data available for any given area. In other words, it represents a query covering the entire globe at highest resolution of data points in space and time for all types of sensor data in the system.

The dimensions of the space to be indexed are assumed to be numbered from 1 to d. In this specific case, dimensions 1 and 2 represent the spatial dimensions x and y of the query region. Additional dimensions can be used to represent attribute ranges. However, to clarify the process of indexing, we restrict ourselves to these two dimensions for now.

A query region is now indexed as follows: Starting at dimension with index 1, the first digit of the DZ expression is determined depending on the extent of the region in the direction of x. If the query region is located entirely in the lower half of the dimension, a 0 is appended to the expression. Similarly, if the region is located entirely in the upper half, a 1 is appended. The process is then continued in the corresponding subspace for dimension 2. Depending on the extent of the query in direction of y, a 0 or 1 is appended to the expression. Alternating between the two dimensions, the expression is extended as long as the query fits entirely into the lower or upper half of the remaining range of x or y, respectively. The final DZ expression is obtained when the region can no longer be assigned to a single half in either dimension which is denoted by *. A more compact representation of multiple expressions can be obtained by introducing asterisks as a placeholder for either 0 or 1 at any given digit of the expression as depicted in Figure 2.

Based on their DZ expressions $dz(q_1)$ and $dz(q_2)$, two queries q_1 and q_2 can now easily be checked for containment. Let l_1 and l_2 be the length of the two respective expressions. If $l_1 \leq l_2$, the first l_1 bits of the two expressions have to match as follows to make sure q_1 contains q_2: For each position in the expression, either the character at that position of $dz(q_1)$ is * or it is identical to that of $dz(q_2)$. If the compact representation is not used, an additional, test can be used to speed up the check: If $l_1 > l_2$, q_1 cannot contain q_2, since a longer expression always represents a smaller region as can be seen from the construction process.

As stated earlier, additional dimensions can be integrated into the index. Instead of alternating between dimensions 1 and 2, the digits in the DZ expression are derived in a round robin fashion for all d dimensions. After processing dimension d, the indexing continues again with dimension 1. The next section describes how we use this property to integrate resolution information in the indexing scheme.

4.2 Supporting Multi-Resolution Mapping

While the previous section described the indexing procedure, we now focus on the extensions needed to support multi-dimensional queries. Without further extension to the index the only way to provide sensor data at multiple resolutions would be to index every data point separately. This, however, would come along with a huge overhead with respect to the number of queries which need to be maintained as well as the length of their representation. Consider we double the resolution addressable by the system. The system would now need to maintain four times as many queries as well as an additional digit for representing each DZ expression. In general, the overhead would grow exponentially as the system is scaled up to a higher resolution.

Instead, the approach taken in this paper incorporates the resolution information directly into the index. This way, the number of queries remains independent of the maximum resolution provided by the system.

4.2.1 Index Extension

The extended index uses additional dimensions for space and time of the resolution. More specifically, dimensions 3 and 4 correspond to the resolutions r_{xy} and r_t of the query. By defining constraints on the intervals of these additional attributes, the clients request a lower resolution of the region of interest. For instance, by constraining a query to one quarter of the dimension representing the spatial resolution, only every fourth data point is delivered to a client in the resulting data stream. Furthermore, by combining two distinct intervals of low resolution the total resolution can be increased without requesting a completely new data stream. If a client requested the lowest quarter of the temporal dimension using one query it may later request the second lowest quarter in another. The client can combine the two resulting data streams very efficiently to get half of the maximum possible temporal resolution. This allows to serve a high resolution request of a user by providing two data streams of lower resolution which provides higher flexibility in managing data streams.

Note, however, that adding resolution information on additional dimensions requires a mapping of data points. If an additional dimension for resolution is added to the GBD-tree without further adaptation, a DZ expression that covers half of the spatial resolution would result in only covering half of the queried area. The following example will illustrate this problem: Suppose we only have a single type of sensors and only consider spatial resolution information in a flat (2D) space. As a result, three dimensions would be indexed with the GBD-Tree, one for each spatial dimension and an additional for resolution. When evaluating a DZ expression, the query region is split in half with every digit of the expression, according to the indexing mechanism described in Section 4.1. Depending on whether the digit is 0 or 1, the lower or upper half of the region is selected by the query. However, when evaluating the resolution dimension, every second point in the region should be selected instead of limiting the queried region.

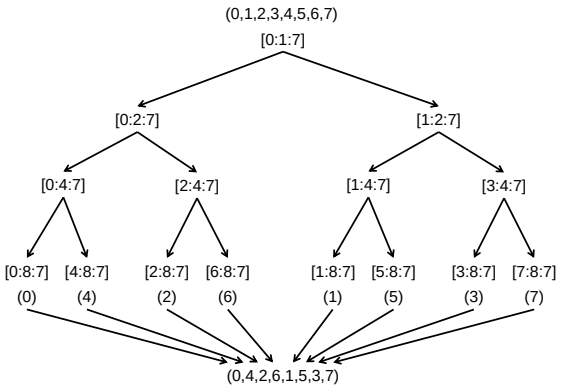

Figure 3: Example for the permutation of indices between 0 and 7.

To achieve the desired properties, the mapping of data points to the additional dimensions needs to be adapted. A mapping has to be found which orders data points in a way that selecting any contiguous block of the reordered data does not limit the boundaries of the selected region. In order to distinguish between the two types of constraints and provide an efficient processing mechanism, the mapping is applied to blocks of data. Each of these blocks is reordered and the constraints of the resolution dimensions are applied to all blocks covered by a query.

Recall, that resolution and region constraints are indexed round robin. We use the same maximum index depth k for region and resolution constraints in our system and, hence, the number of blocks is equal to the number of data points per block. The complete range of points for a single dimension is divided into $m = 2^k$ blocks of m points. A total of $m^2 = 2^{2k}$ points per dimension can therefore be managed for a given value of k. For each block, the respective data points are permuted and mapped to the additional dimension.

As a result of this approach, only the subset which corresponds to the requested resolution is included in a query response. An even distribution of data points over the whole query area is ensured by permuting data point indices before mapping them to the resolution dimension. The permutation scheme is described in the following section.

4.2.2 Permutation Scheme

The order required for a bottom up evaluation of data points is provided by a recursive permutation of elements. An example for the scheme is given in Figure 3. The basic idea behind the permutation is that if a user requests one half of the resolution dimension, every second data object should be delivered.

To explain the generation of the permutation we define the interval $[l : s : u]$ as follows:

$$[l : s : u] = \{n : l \leq n \leq u \wedge n = k \cdot s + l \text{ with } k, n \in \mathbb{N}\}$$

In other words, $[l : s : u]$ describes each s^{th} numbers starting at l and being smaller than u. With m being the size of one block in our scheme, a complete block is therefore represented by $[0 : 1 : m]$. Following the basic idea, this interval is split into two intervals $[0 : 2 : m]$ and $[1 : 2 : m]$. In the second step, the first interval is now split into $[0 : 4 : m]$ and $[2 : 4 : m]$. Similarly, the second interval is split into $[1 : 4 : m]$ and $[3 : 4 : m]$. For each of the subintervals, the process is repeated k times in total. After step i, the first interval always contains every 2^{ith} index, starting at 0. Concatenating this interval and the second one results in the doubled number of indices while still covering the area of the original interval. This way, contiguous blocks of the resolution dimension can be combined without the need for further adaptation.

As a result of the splitting, the original interval is divided into m intervals, each containing one single index. The final permutation is obtained by resorting the indices in the order of the intervals in which they are finally classified.

LEMMA 1. *The permutation of blocks can be obtained at linear computational cost, $\mathcal{O}(n)$, with respect to a given block size n.*

PROOF. The permuted order of elements can be determined in a non recursive process: The binary representation of the original index equals the binary representation of the permuted position in reversed order. As we have to inverse n elements, the computational cost is in $\mathcal{O}(n)$ assuming that a bit sequence can be inverted at constant cost. □

By using this direct computation, the permutation of indices can be done in a highly efficient manner. Consequently, we do not take the computational effort for creating the permutation into account throughout the rest of the paper.

4.3 Query Processing

A user query q can be accepted by any broker, say b, of the system. The broker first generates the corresponding DZ expression $dz(q)$ of the query. Afterwards, the broker checks whether the queried area is available locally, i.e. which parts of it are covered by the set of locally available regions L. The available subqueries are added to the set of client queries C. The part(s) of the query which cannot be answered locally are forwarded towards the broker which is responsible for the respective region. All regions in L and C are maintained in a local GBD-Tree.

4.3.1 Local Processing

Initially, the set of locally available regions L, contains all regions for which data is gathered at the local broker. Over time, further regions are added to L as neighboring brokers replicate certain regions for load-balancing. As a new query is received by a broker, it checks the local availability of the data for that query. A formal description of the function *serve* is given in Algorithm 1. The function operates on the local GBD-Tree that stores L.

As the tree provides a strict containment relation between parent and child nodes, the availability of data for a particular query can be verified very efficiently. Each node in the tree may have zero, one, or two children. The process starts at the root node, which contains the largest possible region and, therefore, also q. If the node is empty, the child node to check next is chosen according to the first digit of $dz(q)$. If the digit is 0, *lowChild* is checked next, otherwise the search continues on *upChild*. As long as the nodes are empty, i.e. no query is stored in that node, this process continues. The next node to check is always selected according to the next digit of $dz(q)$. Eventually, there is no additional digit left to choose the next child node which means that the query region is not entirely available locally.

If a node is nonempty during the process, the data is available and the process ends immediately. The query is inserted into the GBD-tree that maintains C similar to the availability check in L. However, child nodes are created where they do not yet exist. After the last digit of $dz(q)$ has been parsed, the query is inserted in the tree.

In case the entire query region is not available at the broker, the tree is traversed further down to find available subregions. Subregions are available if the node corresponding to the new query has any child nodes. The new query is then split in two subqueries and both are checked for availability in the two corresponding subtrees. Eventually, a set of DZ expressions Q is obtained which describes all fragments of the new query that are not available at the local broker. Formally, $Q = \{q' : q' \notin L \wedge q' \leq q\}$.

4.3.2 Query Routing

For each $q \in Q$, b forwards q to the neighbor *target* whose area is geographically closest to the center of the area represented by q. The neighbor which receives the query performs the same local query processing as described earlier. As the process continues, the query is subsequently forwarded towards the broker that initially provides the required data. Eventually, if the data cannot be provided earlier, the query arrives at that broker and is then served.

To determine which neighbor is closest to the target, each broker maintains a list of regions adjacent to the locally maintained region along with the broker responsible for each of them. Depending on the type of sensor queried, the according neighbors are selected as candidates for forwarding. The information of the spatial extent of the query is then used to determine the broker which is responsible for the region closest to the query region. Algorithm 2 formally describes the query routing. More advanced approaches to this problem, like CAN[20], exist. However, as most traffic is caused the query results rather than queries, the focus of this work is on the indexing and replication mechanisms.

The data for subregions which are locally available is directly delivered to the client who posed the query to the system. Eventually, each query part is forwarded to a broker which has a replica of the data available or the broker responsible for generating the data.

Algorithm 1 Local processing algorithm

Ensure: $Q = \{q' : q' \notin L \wedge q' \leq q\}$

```
1: function serve(inout Set Q, in Query q)
2: if not this.queryList.isEmpty() then // q ∈ L
3:    C.add(q) // add to client requests
4: else if lowChild == null and upChild == null then
   // q ∉ L
5:    Q.add(q)
6: else if dz(q).length() > this.level then // availability
   not clear
7:    if dz(q).getCharAt(level) == 0 then // decide on
      further path
8:       if lowChild == null then // q ∉ L
9:          Q.add(q)
10:      else // continue on child node
11:         lowChild.serve(Q,q)
12:      end if
13:   else
14:      if lowChild == null then // q ∉ L
15:         Q.add(q)
16:      else // continue on child node
17:         upChild.serve(Q,q)
18:      end if
19:   end if
20: else // query unavailable as a whole, check parts
21:   if lowChild == null then // lower half unavailable
22:      Q.add(q.splitLow())
23:   else // continue on child node
24:      lowChild.serve(Q,q.splitLow())
25:   end if
26:   if upChild == null then // upper half unavailable
27:      Q.add(q.splitUp())
28:   else // continue on child node
29:      upChild.serve(Q,q.splitUp())
30:   end if
31: end if
```

Algorithm 2 Query routing algorithm

Require: $Q = \{q : q \notin L\}$ set of query parts to forward

```
1: function forward(in Set Q)
2: Broker target = null
3: Float oldDistance = 0
4: Float newDistance = 0
5: for q ∈ Q do // process all queries in set
6:    oldDistance = getDistance(reg_q,this)
7:    for n ∈ getNeighbors(A_q) do // select neighbors with
      requested data
8:       newDistance = getDistance(reg_q,n)
9:       if newDistance < oldDistance then
10:         oldDistance = newDistance
11:         target = n
12:      end if
13:   end for
14:   send(target,q)
15: end for
```

If many queries lie in the area of a single broker, the bandwidth resources of this broker might be insufficient for serving all incoming queries. A replication of parts of the area for which the broker is responsible therefore has to be established. Another broker can then serve a number of queries for the replicated region, thereby distributing the load to more than one broker. Afterwards, the query is either forwarded along the new outgoing data streams or answered locally, if enough resources have been freed. The procedure of choosing the region to replicate and selecting the broker on which the replica should be established is described in the next section.

4.4 Load Balancing

The load balancing mechanism of the presented approach is directly integrated into the query processing. During query processing, queries are eventually answered by a replicate of the queried area before they are even forwarded to the original source. This way, brokers can be effectively alleviated from high query load on small regions managed by a single broker. If a broker runs out of resources to answer new queries, the replication mechanism establishes a replica of a highly loaded region on a neighboring broker. By selecting neighbors as nodes for replications we ensure that replicas are used to answer a query before it is forwarded to the original source.

When establishing a new replica for load-balancing, two main factors have to be considered: which region is replicated and on which node is the replica established. The selection of the region to replicate is supported by the structured storage of queries in a GBD-tree. By maintaining only a small amount of additional meta-data in the tree, a high flexibility of the metrics for selecting a certain region can be provided. In the following, the details of the replication region selection mechanism are presented. Afterwards, the selection of a node for the new replica for possibly high system robustness is described.

4.4.1 Replication Region Selection

The selection of the region to replicate has significant influence on the performance of the load balancing. The intuition behind our scheme is to replicate parts of the query with high load. In this case, the load of an area is determined by the amount of data that needs to be provided to clients divided by the size of that particular area.

With the GBD-tree maintained at each broker, all regions for which data is available at a broker can be efficiently monitored. For each subregion of arbitrary size, load distribution with respect to the number of clients and bandwidth required to serve clients can be easily obtained. In each node of the tree, the total query size for the entire subtree is updated each time a query is added to or removed from the tree.

The selection process checks for each node in the tree the load of the corresponding covered queries by parsing the local tree of client queries. Formally, the load of a region r is defined as

$$load(r) = \sum_{\{q:q \in C \wedge q \leq r\}} |q|/|r|$$

A region r has potential overload if the total bandwidth required to serve queries in the corresponding subtree is larger than the covered region, e.g. $load(r) > 1$. In other words, if some or part of the data from a certain region has to be sent out more than once, the region has a certain overload. A broker can reduce the local load by only sending out a single copy of the overloaded region to another broker, thereby delegating the task of distributing the data.

If such an overloaded region is found in the local client query tree, it is added to the list of candidates for replication. Further regions are then checked until all candidates are found. Finally, the list of candidates is sorted according to the overhead detected to ensure that highly loaded regions are replicated first. The search process for overloaded regions is given in Algorithm 3.

Algorithm 3 Region candidate selection algorithm

```
 1: function find(out Set R)
 2: if not this.queryList.isEmpty() then // only replicate
    requested data
 3:    if lowChild != null or upChild != null then // po-
       tential for reducing load
 4:       C.add(queryList)
 5:    end if
 6: else // check children
 7:    if lowChild != null then
 8:       lowChild.find(R)
 9:    end if
10:    if upChild != null then
11:       upChild.find(R)
12:    end if
13: end if
```

A replication request is sent to another broker for each of the regions, starting at the one with the largest overhead. If a replica is successfully established, the according client queries are relocated to the newly established replica. As soon as enough bandwidth has been freed by offloading queries to the newly created replica, the client query which triggered the replication is finally processed.

To avoid frequent reorganization of replicas, we restrict the size of a region which is replicated to a certain minimum. This is done by parsing the index tree only down to a maximum level thereby replicating a larger region even if not the entire region is currently queried. The replicated region therefore can also contain parts which are not highly loaded as described above. However, any replicated region r still satisfies $load(r) > 1$ as the load for other partial regions is accordingly higher. Although this introduces additional overhead for the replication of regions, the replicas can then directly respond to requests in the proximity of a highly loaded region.

The replicated region does not need to originate from a broker but can also be a replication itself. New replicas for highly loaded subregions can therefore also be established on more distant brokers as second or even higher level replicas. This way, the system also efficiently prevents denial of service attacks where many users query a small region on a single broker.

4.4.2 Replication Node Selection

As described earlier, a new replica is established on a broker which is responsible for a neighboring spatial region. During query processing, a replica can therefore be used to answer a query before the original, overloaded, broker needs to deliver any additional data. For establishing a new replica, the most intuitive approach is to select the neighbor with the highest amount of available bandwidth. That neighbor will most probably be able to serve the replica without running into an overload situation. Additionally, with this selection strategy, collisions of replication requests can be avoided in presence of multiple query hot spots in close proximity. As the load is propagated to the surrounding regions of the hot spot, two hot spot neighborhoods will eventually reach adjacent brokers. In this case, the new replicas are established in other directions rather than inside the neighborhood of another hot spot as the load is already high there.

However, not all neighboring brokers are considered as candidate for establishing a new replica. Brokers, which already provide data to the local broker are excluded from the candidates. If two neighboring brokers bidirectionally exchange replicated data, the system would otherwise be busy replicating data instead of serving queries. Similar to approaches for application layer multicast, only neighbors that are responsible for a region which has a larger distance to the original region are considered for hosting a replication. By distributing the replicas away from the original query hot spot, circular replication requests can be avoided and a directed distribution of data is ensured.

4.4.3 Removal of Queries

When a client is no longer interested in the data of a query, it sends a message to the brokers which serve the data. If the broker only acts as a relay and does not have any other outgoing connections to serve all or part of the affected area, the cancel message is further propagated. All brokers that provide parts of the no longer required area are informed about the removal of the query. Additionally, relay brokers notify the according providers if they are no longer required as a relay if there is only a single outgoing query left. This way, unnecessary relays can be removed from the data distribution network.

4.5 Properties

The presented combination of query routing and replication strategy provides the desired properties for a global sensing system: equal distribution of load among brokers with efficient handling of queries. The balanced distribution of load is ensured using the information obtained from the GBD-tree which is in turn used to manage client queries. A broker replicates regions of high interest to a broker which is responsible for a neighboring spatial region.

Together with the proposed query processing algorithm, the replication strategy provides a high probability that a replicate of the region of most interest will be contacted by the query routing before putting additional load on the overloaded originating broker. As replicas can be established over multiple levels, even very small query hot spot load can be distributed over a large number of brokers. This way, all system resources can be used to serve user queries which greatly improves the scalability of the entire system. At the

same time, the system efficiently respond to attempted denial of service attacks where many users query a small region on a single broker.

A low overhead for the replication is ensured by the additional matching of resolutions of interest instead of replicating certain regions at the highest possible resolution. As shown in Section 5, our approach can reduce the replication overhead by more than 40%. Our proposed extensions to the indexing provide an efficient way to identify the spatial regions and the resolution of interest. This information is then used by our replication and query processing algorithm to provide multi-resolution query support in a global sensing system.

4.5.1 Required Storage Space

The maintenance of the underlying index on the broker only requires a small amount of storage. Three separate GBD-Trees are used for each sensor type to store data for incoming and outgoing data streams from and to other brokers as well as client requests served. The number of connections to and from other brokers is usually two orders of magnitude smaller than the number of client connections. As only nonempty nodes are stored in the GBD-Tree, the space required for the indexing of replication data streams can be neglected.

Each node in the GBD-Tree stores a list of queries with their corresponding DZ expressions and two pointers to its children. Queries are represented by the lower and upper bound on each of the four dimensions (latitude, longitude, spatial resolution, and temporal resolution). To identify the client (or corresponding source/destination broker, respectively) a network address is stored for each query. The length of a DZ expression corresponds to the level in the tree where it is stored. Since DZ expressions are represented by binary strings, they can be stored as a single integer. Therefore the expressions only slightly increases the storage size required for a query.

4.5.2 Query Processing Cost

Apart from the required storage, the extension of the index also has influence on the query processing and the connections between brokers. As indexing is adjusted more precisely to the query regions, smaller fragments are replicated between brokers. Therefore, the number of connections increases with a constant factor compared to the approach without further extension. However, the number of connections can be reduced in exchange for higher bandwidth usage. Overall, our approach outperforms the single resolution approach with respect to bandwidth usage when limited to the same number of connections.

The proposed query processing only checks for a node in the tree whether there is a query or not. Therefore, the performance of the query processing mostly depends on the number of subregions which comprise the entire data available at the broker. In other words, the time required for query processing increases with the number of replications that are hosted at a particular broker. The search for candidate regions for replication also depends on the total number of distinct regions in the index rather than the number of queries. Therefore, the approach scales well with the number of queries posed to the system.

Another scalability aspect is the number of brokers supported by our approach. As more brokers are integrated, the spatial regions assigned to each broker become smaller. The assignment of smaller areas to brokers leads to an exponential increase in the number of regions representing the areas as they are also indexed for higher spatial and temporal resolutions during the process. However, to support larger numbers of brokers, the top levels of the GBD-Tree index can also omit the resolution dimensions so that the number of regions scales proportional to their size. With this mechanism, the granularity of spatial and resolution indexing can be balanced according to the requirements posed to the system.

5. EVALUATION

The proposed system was implemented using the PeerSim network simulator [16]. 16 brokers were simulated where the entire area was equally distributed among all brokers. Each broker has enough bandwidth resources to serve 25 times the entire region for which it is responsible. To show the ability of the presented system to efficiently distribute high query load in a small region among brokers, a Query hot spot was modeled. Using a zipfian distribution for the lower and upper bounds of the queries, we generated a set of queries of varying size. Following the distribution, most of the queries cover a small area around the hot spot, while few queries still cover larger areas. The size of the hot spot was modified by choosing different exponents for the zipfian distribution. The center of the hot spot was placed in the center of the overall area. Values for both, temporal and spatial, resolutions were chosen from a uniform distribution.

Queries were posed to the system until it was unable to serve any more requests, i.e. no suitable configuration could be found to accommodate the new request. The focus of the evaluation was to show the ability to distribute the load among brokers in presence of query hot spots. The cost of maintaining the replicas in terms of connections and messages that are needed to establish the connections has been investigated. Our results show that indexing the resolution information of queries can significantly reduce the overhead required for replication. In addition, the maintenance cost can be flexibly constrained. By limiting the indexing depth of the GBD-tree the replicated regions can be extended in a controllable way.

5.1 Required Connections and Messages

Figures 4 and 5 shows the average number of connections per broker that have been established during the experiment to maintain the replicas for a large and small hot spot area, respectively. The numbers shown do not include connections to clients but only the connections required for delivering data inside the broker network. The results clearly show that a finer indexing of all dimensions requires far more connections to be maintained. This is due to the fact, that queries are split into smaller subqueries to achieve a better approximation of the actually queried region.

In turn, the number of messages exchanged during the establishment of these connections is much higher for a larger

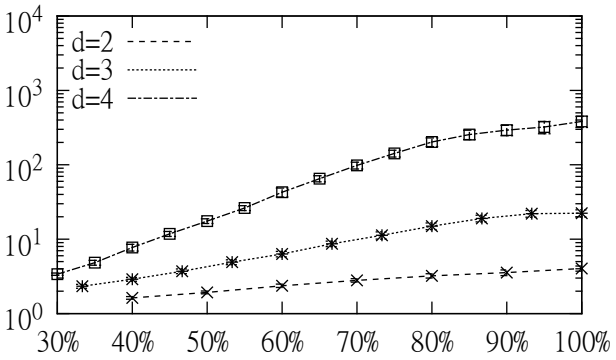

Figure 4: Number of connections required w.r.t. index depth for a large hot spot.

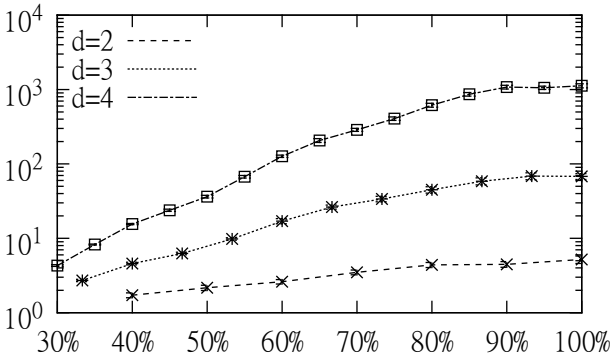

Figure 5: Number of connections required w.r.t. index depth for a small hot spot.

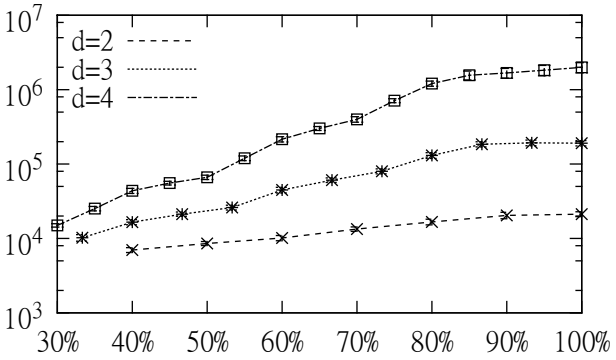

Figure 6: Number of messages required w.r.t. index depth for a large hot spot.

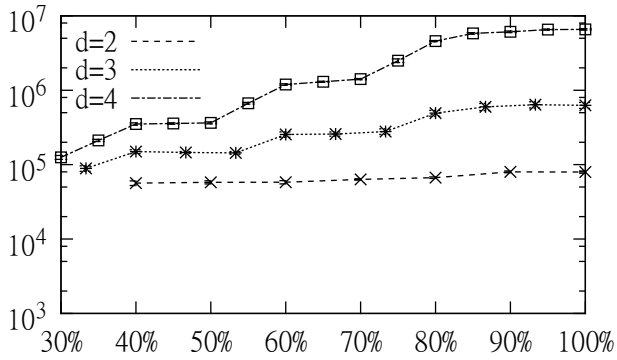

Figure 7: Number of messages required w.r.t. index depth for a small hot spot.

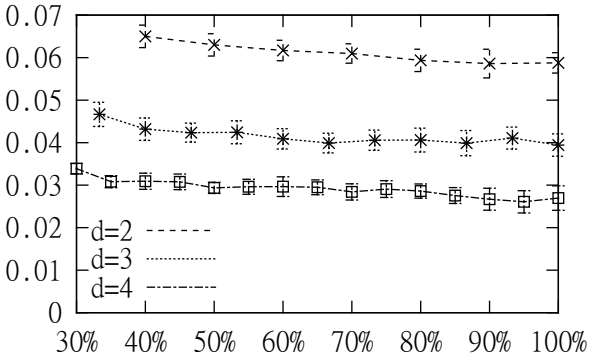

Figure 8: Percentage of bandwidth required for the replication w.r.t. index depth for a large hot spot.

indexing depth as depicted in Figures 6 and 7. The numbers only include messages required for establishing, modifying, and removing connections, not the delivery of payload. We can also observe that the number of connections and messages is larger for smaller hot spot areas. This is due to the fact that the smaller areas are matched more closely by our replication algorithm.

5.2 Resource Consumption of Replication

As shown in Figure 8, the required overhead can be significantly reduced by increasing the number of indexed dimensions in case of large hot spot areas. This reduction can save over 40% of the payload traffic compared to the pure spatial indexing without resolution information. A similar gain can also be achieved for small hot spots as shown in Figure 9. However, the graph also indicates that the indexing depth is an important factor for tuning the system. The resource consumption for the replication mechanism can be greatly reduced using a higher indexing depth.

5.3 Total Achievable Query Load

The cost of decreased overhead is not only the increased number of connections and messages as described in the previous sections, but also in the stability of the system. Fig-

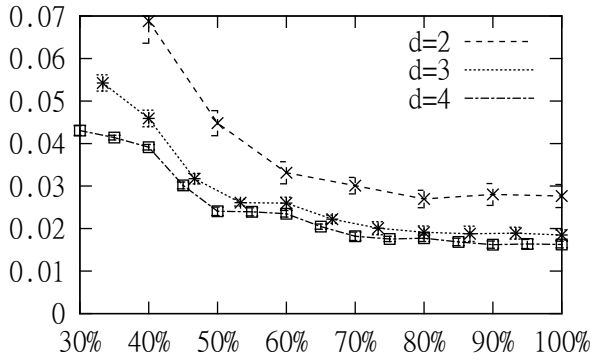

Figure 9: Percentage of bandwidth required for the replication w.r.t. index depth for a small hot spot.

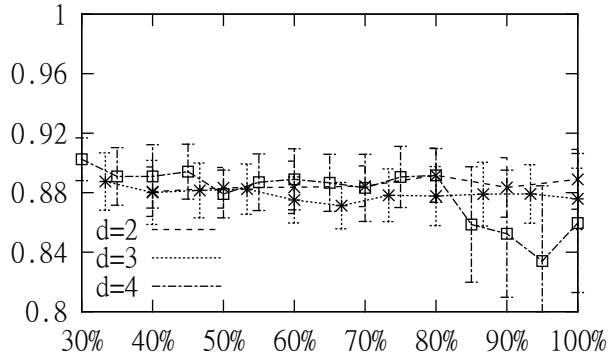

Figure 10: Maximum system load that can be served w.r.t. index depth for a large hot spot.

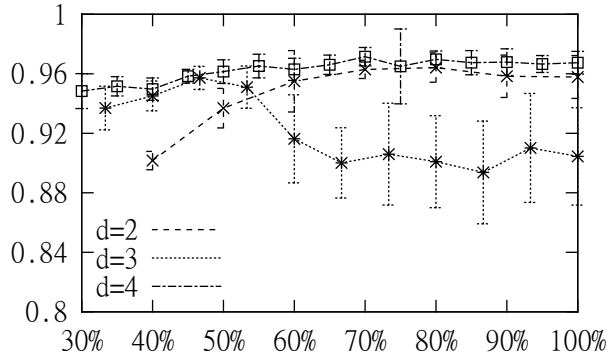

Figure 11: Maximum system load that can be served w.r.t. index depth for a small hot spot.

ure 10 shows the maximum achievable load for the system with varied indexing depth for the large hot spot scenario. The graph shows how far the system could be loaded with queries until no suitable distribution of data streams could be found to answer the new query.

As the larger hot spot areas cover more than a single broker, the resolution of conflicts for different replication requests becomes difficult with many small partial queries. Therefore, the results for a high indexing depth show a larger standard deviation indicating that the maximum load is slightly lower for certain experiments. When only indexing one additional dimension, the system can resolve the conflicts more reliably and therefore serve more data in total for high indexing depths.

For the smaller hot spot regions, as shown in Figure 11, this effect does not occur since they are usually covered by a single broker. In this case, the replications can be established on any neighboring broker as they do not need to establish replicas themselves. Figure 11 also indicates that adding both spatial and temporal resolution to the index results in a high achievable system load over a broad range of indexing depths. For only two indexed dimensions the replication overhead becomes too large when using a low indexing depth. When adding only the spatial resolution dimension, the replication overhead is significantly reduced. However, as requests for low temporal and high spatial resolution increase, the system cannot expand to its full potential. Overall, both graphs show that our approach can distribute the load effectively among all broker in the system and therefore provide a high total client load.

6. CONCLUSION AND FUTURE WORK

We have described a system for scalable and efficient distribution of grid-based sensor data. The system is capable of providing data at multiple resolutions at the same time without transmitting redundant data streams. To achieve this, we extended the GBD-Tree to allow for the indexing of resolution information. We also provided the required mechanisms to allow efficient query processing while incorporating the resolution data. As a result, the bandwidth available in the entire broker network can be used in a very efficient fashion by the proposed system.

Our load distribution approach alleviates single brokers from their load and thereby remove bottlenecks when query hot spots occur. The proposed replication approach avoids high overhead through full replication by only replicating data at required resolution. The indexing based on the GBD-Tree is also scalable to a high number of brokers as each broker can locally generate the DZ expression for a query and there is no need for a centralized organization.

In the future, we want to further improve and extend the region selection algorithm to anticipate moving hot spots. Such moving hot spots might occur if many users monitor the environment of environmental phenomena like hurricanes, for example. To achieve further improvements in efficiency, the system will also be extended using adapted compression techniques. In addition, the discovery and management of neighboring nodes will be improved by integrating more advanced structured overlay approaches.

7. ACKNOWLEDGMENTS

The author A.B. would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

8. REFERENCES

[1] D. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, et al. The design of the borealis stream processing engine. In *CIDR '05: Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research*, Asilomar, California, 2005.

[2] K. Aberer, M. Hauswirth, and A. Salehi. The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical Report 006, EFPL, 2006.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.

[4] A. Benzing, B. Koldehofe, M. Völz, and K. Rothermel. Multilevel predictions for the aggregation of data in global sensor networks. In *DS-RT '10: Proceedings of the 14th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, pages 169–178. IEEE, 2010.

[5] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed processing of very large datasets with datacutter. *Parallel Computing*, 27(11):1457–1478, 2001.

[6] J. A. Briones, B. Koldehofe, and K. Rothermel. Spine : Adaptive publish/subscribe for wireless mesh networks. *Studia Informatika Universalis*, 7(3):320–353, 2009.

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. on Computer Systems*, 19(3):332–383, 2001.

[8] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. *SIGOPS Oper. Syst. Rev.*, 37:298–313, October 2003.

[9] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[10] A. Chervenak, E. Deelman, C. Kesselman, B. Allcock, I. Foster, V. Nefedova, J. Lee, A. Sim, A. Shoshani, B. Drach, D. Williams, and D. Middleton. High-performance remote access to climate simulation data: a challenge problem for data grid technologies. *Parallel Computing*, 29(10):1335–1356, 2003.

[11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[12] M. Franklin, S. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design considerations for high fan-in systems: The hifi approach. In *CIDR '05: Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research*, Asilomar, California, 2005.

[13] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[14] C. Gebhardt, S. Theis, M. Paulat, and Z. B. Bouallègue. Uncertainties in cosmo-de precipitation forecasts introduced by model perturbations and variation of lateral boundaries. *Atmos. Res.*, 2011.

[15] P. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. Irisnet: An architecture for a worldwide sensor web. *IEEE Pervasive Computing*, 2(4):22–33, 2003.

[16] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[17] Y. Ohsawa and M. Sakauchi. A new tree type data structure with homogeneous nodes suitable for a very large spatial database. In *ICDE '90: Proceedings of the Sixth International Conference on Data Engineering*, pages 296 –303, 1990.

[18] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 563–574, New York, NY, USA, 2003. ACM.

[19] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, page 49, Washington, DC, USA, 2006. IEEE Computer Society.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[21] S. Rizou, F. Dürr, and K. Rothermel. Solving the multi-operator placement problem in large-scale operator networks. In *ICCCN 2010: Proceedings of the 19th International Conference on Computer Communication Networks*, Zurich, 2010. IEEE Communications Society.

[22] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, and M. Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. Technical Report TR-21-04, Harvard University, 2004.

[23] M. Tariq, G. Koch, B. Koldehofe, I. Khan, and K. Rothermel. Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints. In P. D'Ambra, M. Guarracino, and D. Talia, editors, *Euro-Par 2010 - Parallel Processing*, volume 6271 of *Lecture Notes in Computer Science*, pages 458–470. Springer Berlin / Heidelberg, 2010.

[24] M. A. Tariq, B. Koldehofe, G. G. Koch, and K. Rothermel. Providing probabilistic latency bounds for dynamic publish/subscribe systems. In *KiVS '09: Proceedings of the 16th ITG/GI Conference on Kommunikation in Verteilten Systemen*, Kassel, Germany, 2009. Springer.