StreamGlobe: Adaptive Query Processing and Optimization in Streaming P2P Environments

Bernhard Stegmaier Richard Kuntschke Alfons Kemper

TU München - Lehrstuhl Informatik III
Boltzmannstraße 3
D-85748 Garching bei München
Germany
<*first_name.last_name*>@in.tum.de

Abstract

Recent research and development efforts show the increasing importance of processing data streams, not only in the context of sensor networks, but also in information retrieval networks. With the advent of various mobile devices being able to participate in ubiquitous (wireless) networks, a major challenge is to develop data stream management systems (DSMS) for information retrieval in such networks. In this paper, we present the architecture of our *StreamGlobe* system, which is focused on meeting the challenges of efficiently querying data streams in an ad-hoc network environment. StreamGlobe is based on a federation of heterogeneous peers ranging from small, possibly mobile devices to stationary servers. On this foundation, self-organizing network optimization and expressive in-network query processing capabilities enable powerful information processing and retrieval. Data streams in StreamGlobe are represented in XML and queried using XQuery. We report on our ongoing implementation effort and briefly show our research agenda.

1 Introduction

In recent years, Peer-to-Peer (P2P) networks have gained huge attention both in the media and the computer science community. This is, on the one hand, due to the stunning success of filesharing systems like, e.g., Napster and Gnutella. But on the other hand, it is also caused by the degree of flexibility these networks provide. For example, they can be used for setting up ad-hoc sensor networks where sensors can join and leave the network at any time, e.g., while moving across the area covered by the respective network. Of course, this does not only hold for the data delivering sensors, but also for the network nodes that query the data streams within the ad-hoc network. In the past, various approaches for finding information, i.e., documents, files, etc., in P2P networks have been studied, which has led to a number of topologies for P2P networks, one example being super-peer networks [28]. Dealing with data streams, finding peers which deliver the required information is not the only task. Additionally, a continuous data flow from data sources to consumers in the network has to be established. An interesting challenge arising in this highly dynamic environment is to develop a distributed, self-organizing system for efficient routing and in-network query processing. We pursue this goal with our *StreamGlobe* system which is based on its predecessor ObjectGlobe [3]. StreamGlobe extends ObjectGlobe—which is mainly focused on distributed query processing for persistent data on the internet—by introducing query processing capabilities on data streams in the network. In our context, data streams are represented in XML and queried (i.e., subscribed) using XQuery. While StreamGlobe is not restricted to sensor networks, we use them as a motivating example in the following.

Consider Figure 1 as an abstract example of a possible application scenario for StreamGlobe. The depicted network contains four so-called *super-peers* (SP_0 to SP_3), forming a stationary super-peer backbone network, and five possibly mobile *thin-peers*, or peers for short, (P_0 to P_4) connected to the backbone. Peers P_0, P_2 and P_3 are a cell phone, a laptop, and a PDA, respectively. These peers are meant to register queries in the network and are therefore at the receiving end of data streams. In contrast to that, peers P_1 and P_4 are sensors delivering their sensor data to the network in the form of XML data streams. Two examples for applications of similar real-life networks would be satellite communication and weather observation. In the former case, orbiting satellites would be the moving sensors—or rather collections of sensors—streaming their data to various receiving stations on the ground for evalu-

Proceedings of the First Workshop on Data Management for Sensor Networks (DMSN 2004),
Toronto, Canada, August 30th, 2004.
`http://db.cs.pitt.edu/dmsn04/`

ation. In the latter case, the sensors would be attached to weather balloons or observation planes, delivering data like temperature, humidity, etc. to enable weather forecasts for different regions.

To illustrate some of the difficulties of query processing in such networks and to motivate our approach, we now introduce a rather simplified real-world example in a little more detail. Let us assume that P_4 in Figure 1 delivers a data stream produced by special sensor suits worn by firefighters in action. The sensors continuously deliver sensor readings containing the corresponding firefighter's identity (`id`), a timestamp (`time`), and the GPS coordinates of the sensor (`x`, `y`), as well as information about the firefighter's vital statistics and the environmental conditions. We have exemplarily chosen to monitor body temperature (`bt`), pulse rate (`pr`), and oxygen saturation (`os`), as well as environmental temperature (`et`), carbon dioxide concentration (`CO2`), and sulfur dioxide concentration (`SO2`). For brevity, we use the following simplified DTD to describe the data stream, although StreamGlobe actually employs XML Schema.

```
<!ELEMENT reading (id, time, x, y,
                   bt, pr, os,
                   et, CO2, SO2)>
<!ELEMENT id (#PCDATA)>
...
```

The remaining elements have analogous DTD entries. Let us now further assume that P_0 and P_2 are devices used by an emergency physician and the fire department, respectively. The former should receive a notification on a cell phone whenever a firefighter's oxygen saturation reaches a critical level. Therefore, the peer represented by the physician's cell phone registers the following XQuery.

```
for $m in stream("firefighters")/reading
where $m/os < 92 or $m/os > 98
return
    <alert>
        {$m/id} {$m/time} {$m/x} {$m/y}
        {$m/os}
    </alert>
```

The fire department wants to monitor the environmental conditions, e.g., to be able to issue a warning if the conditions get critical for the firefighters on site or the residents living nearby. Thus, it registers the following XQuery.

```
for $m in stream("firefighters")/reading
return
    <gas>
        {$m/id} {$m/time} {$m/x} {$m/y}
        {$m/CO2} {$m/SO2}
    </gas>
```

StreamGlobe will handle this scenario as follows. Suppose we want to reduce network traffic. The data of P_4 will be sent to SP_3 where it will be filtered, leaving only the elements `id`, `time`, `x`, `y`, `os`, `CO2` and `SO2` in the stream. The elements `bt`, `pr` and `et` can be removed as they are not

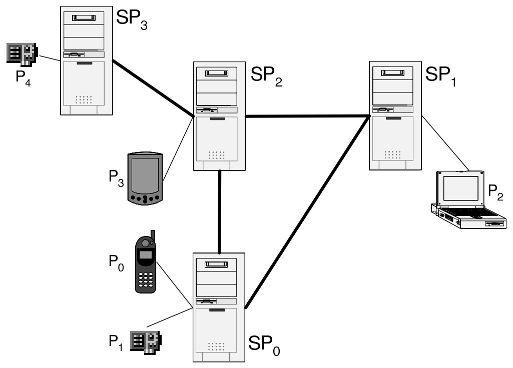

Figure 1: Example Scenario

needed (i.e. not subscribed) anywhere else in the network, leading to a smaller data stream and reducing network traffic. The resulting stream, containing the combined information for satisfying the queries of P_0 and P_2, is routed to SP_2. Note that up to now, data needed by both P_0 and P_2 has been routed as one single stream through the network. At SP_2, however, the stream has to be split into the—in our case—non-disjoint parts for the two receiving peers. This involves replicating the stream and again filtering the two new streams, resulting in two streams which constitute the final results for the two queries. These are eventually routed to P_0 and P_2 via SP_0 and SP_1, respectively.

Decisions such as where to execute which operators in the network and how to route the data streams are made by the StreamGlobe query optimizer. Additional difficulties arise by the fact that the network can change over time by adding or deleting queries and data streams which requires a strategy for continuous or periodic reoptimization. The distinguishing features of StreamGlobe compared to related systems are thereby its self-organizing network, in terms of continuous reactions to dynamic changes in registered data streams and queries, and its routing and optimization approaches for query and network traffic optimization in P2P networks.

The remainder of the paper is organized as follows. Section 2 presents some related work. In Section 3 we give an overview of the StreamGlobe system architecture. Section 4 deals with optimization and query processing in StreamGlobe. In Section 5 we present a brief report on the current implementation status of our StreamGlobe prototype. Finally, Section 6 concludes the paper and gives an outlook on future work.

2 Related Work

In the following, we present an overview of some work related to our StreamGlobe system. In particular, we deal with work in the fields of data stream systems, query processing, network architecture, and grid computing.

2.1 Data Stream Systems

With StreamGlobe being a system that handles and processes data streams, it is worthwhile to take a look at other recent approaches to building data stream systems.

One important project is TelegraphCQ [7]. This is a system that deals with continuously adaptive query processing in a data stream environment. Cougar [30] tasks sensor networks through declarative queries. Aurora [6] is a new DBMS for monitoring applications and constitutes a centralized stream processor for dealing with streaming data. In [10] two complementary large-scale distributed stream processing systems, Aurora* and Medusa, are described. Aurora* is a distributed version of Aurora with nodes belonging to a common administrative domain. Medusa supports the federated operation of several Aurora nodes across administrative boundaries. STREAM [2] incorporates its own declarative query language for continuous queries over data streams and relations. It handles streams by converting them into relations using special windowing operators and converting the query result back into a data stream if necessary. PIPES [20] is a recent public domain infrastructure for processing and exploring data streams.

All of these systems—more or less—focus on special aspects of (adaptive) query processing, load balancing, or quality-of-service management. The major contribution of StreamGlobe is that it does not only efficiently locate and query data streams, but also employs in-network query processing for adaptively optimizing data flow within the network. Thus, StreamGlobe pushes query processing from subscribing clients towards data sources in the network. The optimization is based on data stream clustering derived from clustering the queries in the system. NiagaraCQ [8] intends to achieve a high level of scalability in continuous query processing by grouping continuous queries according to similar structures. In StreamGlobe, we employ a similar multi-query optimization approach to reduce network traffic and to enable efficient query evaluation.

2.2 Query Processing

With respect to query processing, works in the fields of multi-query optimization, as pointed out above, and continuous queries are related to StreamGlobe. Multi-query optimization (MQO) has been addressed in [26]. It pursues the goal of processing multiple queries all at once instead of one query at a time. The main optimization potential lies in the fact that queries may share a considerable amount of common—or at least similar—input data that can be reused for more than one query. Obviously, StreamGlobe in general has to deal with a set of queries simultaneously, thus rendering multi-query optimization an applicable and suitable optimization approach. Also, queries in StreamGlobe are usually continuous queries over data streams. Efficient processing of such queries has been examined in [22]. Query processing in sensor networks has been explicitly addressed in [31].

Multicast in IP, ad-hoc and sensor networks, described for example in [15], routes data towards receiving ends in a way that reduces network traffic by transmitting the same message or document only once for all recipients instead of multiple transmissions, one for each recipient. It is important to point out that our work differs from these approaches in a major way. Instead of only reusing existing messages or documents, our system is able to perform expressive in-network transformations of data streams. Therefore, it can dynamically create appropriate data streams that best fit the queries to be answered while at the same time reducing network traffic.

To achieve this goal, StreamGlobe uses clustering techniques to identify reusable existing data streams in the network that fit newly registered queries. This approach has similarly been applied in the world of persistent data where view materialization and view selection are used to improve the efficiency of query processing [21]. In [29], further algorithms for solving the view materialization problem are devised. Materialized view selection and maintenance have also been examined using techniques of multi-query optimization [23].

As already mentioned, StreamGlobe uses XQuery to query XML data streams. In [11] an XQuery engine called XQRL for processing XQueries on streaming XML data is introduced. In StreamGlobe, we use FluX [19], another XQuery engine for efficiently processing XML data streams. The query containment problem in the context of XML queries, which is relevant for multi-query optimization, has been addressed in [27].

2.3 Network Architecture

Considering network architecture, a lot of work has been done with respect to P2P, Publish&Subscribe, and ad-hoc networks.

P-Grid [1] is a self-organizing, structured P2P system. The notion of self-organization with respect to stream processing and stream routing is also central to StreamGlobe. In [28] the concept of super-peer networks is introduced. These networks are meant to improve the scalability of P2P networks by using a super-peer backbone network. The super-peers usually are powerful servers. Less powerful, possibly mobile thin-peers can register and deregister themselves in the network via the super-peers.

HyperCuP [25] is an approach that uses hypercubes as a network topology in P2P networks. It thereby achieves a logarithmic upper bound for the number of hops needed to get from one super-peer in the network to any other super-peer. This topology is used in [5] to deal with distributed queries and query optimization in P2P systems.

2.4 Grid Computing

StreamGlobe builds on and extends the Open Grid Services Architecture (OGSA) and its reference implementation, the Globus Toolkit [14] by adding data stream processing capabilities to the grid computing domain. A related approach, also building on Globus, is described in [9]. However, this alternative approach concentrates mainly on data stream analysis and quality-of-service aspects in data stream delivery whereas we primarily focus on self-organization, distributed in-network query processing and optimization.

Another system building on the Open Grid Services Architecture is OGSA-DAI (Open Grid Services Architecture Data Access and Integration) [24]. As the name suggests, this project is concerned with constructing a middleware to enable the access and integration of data from distributed data sources via the grid. It also contains a distributed query processor called OGSA-DQP. In contrast to StreamGlobe, OGSA-DAI has no special focus on data streams.

3 StreamGlobe Architecture Overview

StreamGlobe constitutes a federation of servers (i.e., peers) which carry out query processing tasks according to their capabilities. The basic architecture of a peer is depicted in Figure 2. The various layers of this architecture will be sketched in the following. Dashed lines mark layers whose presence depends on the capabilities of the respective peer.

3.1 Open Grid Services Architecture

The StreamGlobe architecture is based on grid standards. Grid computing [13] and the associated Open Grid Services Architecture (OGSA) [12] have gained considerable attention recently. Grid computing denotes a distributed computing infrastructure where computers can exchange data and perform large-scale resource sharing over the grid. To achieve this, an architecture for integrating heterogeneous dynamic services while guaranteeing certain quality-of-service requirements is needed. For this purpose, the Open Grid Services Architecture has been developed.

Despite the growing importance of the grid standards, data stream processing in the grid computing context has hardly been investigated so far. We have decided to implement our StreamGlobe prototype as an extension of the Globus Toolkit for grid computing [14]. Globus is a reference implementation of the Open Grid Services Architecture. Our goal is to use existing Globus techniques for our purposes where possible and to integrate the StreamGlobe system and its functionality into the toolkit as an extension of Globus for data stream processing.

The main aspects of Globus that will be used in StreamGlobe are communication mechanisms and *service data elements*. Service data elements can be associated with any service in the grid. They are essentially XML documents satisfying a given XML Schema and describing properties of the service they are associated with. In our context, service data elements will be used for describing data streams and properties like bandwith of network connections, processing capabilities of peers, etc.

3.2 Network Topology

In the OGSA framework, direct communication between all participating grid services is allowed. However, this behavior is not the normal way of communication in networks including mobile devices. It might not even be desirable in a scenario that tries to reduce network traffic as in our case. For instance, mobile sensors will normally communicate via some kind of access point they are connected to. Hence, in StreamGlobe we establish a logical P2P overlay network constituting a federation of heterogeneous peers. Developing a research platform, we do not restrict ourselves to employing a special P2P network topology for StreamGlobe at the moment. The P2P network consists of a set of *peers*. Each peer has a set of other peers as

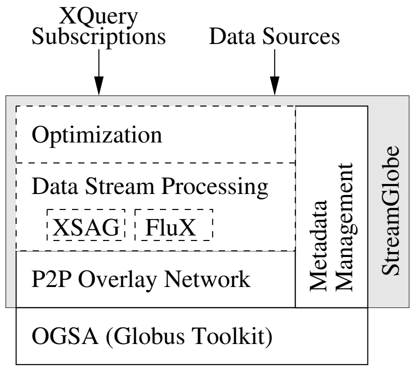

Figure 2: Architecture Overview

neighbors. A peer only interacts with its neighbors, i.e., no direct communication takes place between two peers not being neighbors. If data has to be transferred between two random peers, a *route* between these two peers has to be established such that two successive peers on this route are neighbors and the starting point and the end point of the route are the source peer and the destination peer, respectively. For the implementation of this overlay network, previous work on P2P network topologies can be employed, e.g., a structured approach based on Cayley graphs as used in the HyperCuP [25] topology. Since a major goal is building a network with highly heterogeneous peers with respect to computing power—ranging from small, mobile devices to stationary workstations or servers—, we have to classify peers according to their capabilities. *Thin-peers* are devices with low computational power, like sensor devices, PDAs, cell phones, etc., which are not able to carry out complex query processing tasks. In contrast, *super-peers* are stationary workstations or servers providing enough resources for extensive query processing. These super-peers establish a backbone taking over query processing tasks which cannot be performed by other peers. Thus, they constitute a super-peer backbone network similar to that in [28].

3.3 Client Interface

User interaction in StreamGlobe is depicted at the top layer of Figure 2. StreamGlobe enables clients to specify *subscription rules* for information processing and retrieval using the XQuery language. Subscription rules are registered at certain peers, i.e., normally at the devices users are working with, e.g., their laptops, PDAs, cell phones, etc. In our context, subscriptions are transforming queries and not just queries for retrieving matching files or documents. In fact, StreamGlobe enables expressive transformations of data streams according to registered subscription rules. Thus, it allows clients to flexibly tailor data streams to their individual requirements.

Similarly, data sources also register the provided data streams at a certain peer within the StreamGlobe system. Data streams can be registered in two ways. A data source may register its data stream as an individual stream, which then is published using a unique identifier. Another possibility is registering a data stream as part of a *virtual data* *stream*, which again is accessible using a unique identifier and multiplexes all the data of the participating data sources into one single stream. This technique is used in the introductory example to merge the sensor data of all firefighters. The schema of the data streams is specified using XML Schema. Streams are fed into StreamGlobe using *wrappers*, which are running on corresponding peers and transform the data into a suitable format, e.g., by converting raw sensor data to XML.

3.4 Peer Architecture

A more detailed view of the peer architecture is depicted in Figure 3. It basically reflects the structure sitting on top of the P2P network layer of Figure 2. The various components are implemented as cooperating grid services in the OGSA framework. The individual peers exchange control information, e.g., registration of new neighbors, subscriptions, etc., via a top-level interface service, which dispatches the messages to corresponding subsidiary StreamGlobe services, e.g., the optimization or the query engine service. The communication of these services is conducted via the RPC mechanisms of the Globus Toolkit. All services marked by solid rectangles are mandatory for every peer. Dashed boxes mark services that vary between different peers according to their functionality, as mentioned earlier. For example, thin-peers do not incorporate a complete optimization and query execution unit, but only provide basic functionality. A cell phone might for instance only provide functionality for receiving and displaying data streams and a sensor device might only be able to transmit its measurement data.

The metadata management component, which will be discussed further in the next section, interacts with each of the components and provides information needed for network management, optimization, and query execution. Peers exchange XML data streams representing user data over their data ports. The XML data streams are initially parsed by the wrappers and represented as a sequence of SAX events. Special events are interspersed within these streams which are used for internal purposes. For example, synchronization marks are generated whenever the system restructures the data flow to synchronize all affected peers for the change in query execution. Since the Globus Toolkit currently does not provide suitable techniques for transmitting data streams, we use our own protocol based on TCP connections for this purpose.

3.5 Metadata Management

As Figures 2 and 3 suggest, metadata is needed in all layers of the StreamGlobe architecture. The metadata management (MDV) is based on the distributed metadata management of ObjectGlobe [16] and forms a backbone that peers exchange metadata with. In particular, the metadata management component records the following information:

- **Network:** The metadata management records the neighborhood relationships between peers needed for establishing the P2P overlay network.

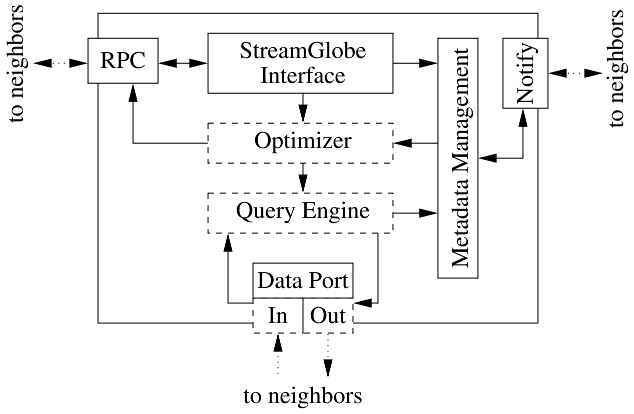

Figure 3: Peer Architecture

- **Subscriptions:** All subscription rules and registered data sources are recorded. For each registered data source, the schema of the data stream is stored. Schemas of data streams are specified using the XML Schema language.
- **Optimization:** The metadata management maintains information needed for optimizing the network. Among others, it maintains properties of network connections, like bandwith and current amount of network traffic. It also maintains the computational capabilities of the peers and statistics of the data streams, i.e., size and cardinality of the elements of a data stream. The statistics can be provided either by the data source itself or by computing them online as the corresponding wrapper feeds the data stream into StreamGlobe.

All metadata is stored locally at a peer in the form of Globus service data elements. For being able to optimize the network, special speaker-peers, which will be introduced in Section 3.6, will need to have more global information about a special set of peers (a certain subnet). In this case, those special peers maintain additional information, e.g., the graph of the network topology of the respective set of peers, or are able to request the desired information from the corresponding peers, e.g., statistics of a certain data stream. To maintain a consistent state, peers have to notify the speaker-peer of changes, e.g., if a peer joins or leaves the network, new subscriptions or data streams are registered or existing subscriptions or data streams are deregistered, etc. Therefore, MDVs of peers register themselves as notification sinks or notification sources at the MDV of their speaker-peer using the notification mechanism of the Globus Toolkit.

3.6 Optimization and Evaluation Strategy

In Section 1, we have briefly introduced our approach of optimizing the data flow in the network using in-network query processing. In the following, we give an overview of the optimization and evaluation strategy we employ in StreamGlobe.

Optimization in a distributed architecture implies several challenges. In order to perform optimization, some metadata about the network—as described in the previous section—has to be available. In a distributed system, there are basically three approaches for performing optimization using such metadata:

1. A single optimizing component has global knowledge of all metadata and performs optimization with a global view of the network.
2. Every peer has only local knowledge of its own metadata (including that its neighbors can be asked for their metadata) and tries to optimize the network by making locally optimal decisions.
3. A hybrid approach, in which special peers have global knowledge of (small) subnets which are individually optimized by the responsible peer.

Since we assume a large, distributed environment, a centralized optimization component as in the first method is infeasible. The second approach fits quite nicely into a distributed P2P network, but it seems unlikely that it will deliver acceptable results. Hence, we focus on the hybrid approach: A selected super-peer, called *speaker-peer*, is responsible for optimizing a certain subnet of the network. Of course, this subnet may include other super-peers that will not actively participate in optimizing this part of the network. With peers joining and leaving subnets, a speaker-peer might decide that a subnet is getting too big (or too small). In this case, the subnet is split into two new subnets and for each new subnet a responsible speaker-peer is elected among the super-peers (or analogously a subnet is merged with a neighboring subnet if it is getting too small). Additionally, by varying the maximum size of a subnet optimized by a speaker-peer, the approaches (1) and (2) can be simulated, which enables an evaluation of all three approaches in terms of optimization quality.

Basically, optimization in StreamGlobe determines the peers at which (at least parts of) the subscriptions are executed and decides how to route the data streams in the network. Optimization has three major goals:

1. Enable users to register arbitrary subscriptions at any (suitable) device regardless of its processing capabilities.
2. Achieve a good distribution of data streams in the network without congesting it with redundant transmissions.
3. Optimize the evaluation of a large number of subscription rules by means of multi-query optimization.

The goals (1) and (3) are accomplished by pushing query execution into the network. Subscription rules, i.e., XQueries, are evaluated using the FluX query engine [19] that was developed in cooperation with our group. The second goal is achieved by placing *filtering operators* on the routes of data streams. These filtering operators are also executed by FluX. They could alternatively be implemented using special filtering techniques such as XSAGs [18]. More details will be presented in Section 4.

Of course, optimization is a continuous process which reoptimizes the system on-the-fly as peers come and go, data sources and subscription rules are registered and deregistered, and data streams change over time.

4 Optimization and Query Processing

In this section, we describe some of our approaches to optimizing network traffic and performing efficient query processing in StreamGlobe. This substantiates the strategy introduced in the previous section.

4.1 Optimization

First, we address the key ideas for achieving the three optimization goals stated at the end of Section 3.6. The first goal is achieved by appropriately pushing subscription evaluation into the network. This is done by executing the subscription as a whole or in part at one or more appropriate peers on a route from the data sources to the peer where the subscription was registered. An appropriate peer is a peer that is able to process the subscription, i.e., has sufficient computing power and is selected by the query optimizer, taking into account optimization goals such as, e.g., reducing network traffic. In order to support powerful subscription rules, the concept of *mobile code* is employed. Besides peers providing a basic set of functionality, users are enabled to include user-defined code in subscription rules, e.g., predicates, aggregation operators, etc. This user-defined code is subsequently instantiated at the peer processing the corresponding part of the subscription.

The second goal is accomplished by using two techniques complementing each other. The first technique is *filtering* of data streams. Filtering is achieved by using either projection (called structural filtering) or selection (called content-based filtering) or both on the elements of a data stream—as described in the example scenario in Section 1—and is performed by *filtering operators*. These filtering operators are executed at peers on the route of the data stream as close to the source of the stream as possible. Thus, the amount of data that has to be transmitted through the network is reduced. The second technique is *data stream clustering*. This term denotes the combination of several similar or equal data streams in the network to form one single stream that serves multiple recipients. Data stream clustering in StreamGlobe works as follows. During the registration of a new query, the system parses the query, identifies its properties and stores them in a suitable data structure. In our case, this will be a Globus service data element. The properties of a query include the data streams needed to answer the query (content aspect), the operations, e.g., projections, selections, joins, etc., used to transform these input streams (structural aspect) and the conditions needed for these operations, e.g., projection attributes, selection and join predicates, etc. All transformed data streams in the system, that where generated by a query, are equally represented by their respective properties. Initial data streams, registered at a super-peer by some data

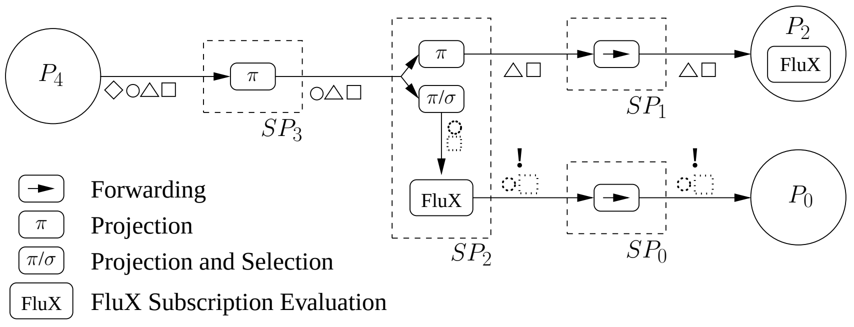

Figure 4: Query Evaluation Plan for the Example Scenario

source, are represented by a unique id. The reason for choosing this properties approach is to get one level of abstraction higher compared to the schema representation of data streams, thus facilitating the comparison of streams and the search for reusable data streams in the network.

During the actual data stream clustering stage, the speaker-peer of the affected subnet looks up all relevant metadata (i.e. service data elements) of existing data streams in its subnet and compares their properties to those of the newly registered query. In a first simple greedy approach, the speaker-peer selects those data streams as input streams for the new query that contain the necessary information for answering the query, contain the least amount of unnecessary information, and have to be routed through the minimum number of peers to get to the recipient. Of course, the decision where to execute certain query operators, e.g., joins, in the network has also to be made. This, along with more sophisticated methods for searching reusable streams and routing them to recipients, is the subject of future research and will be based on an appropriate cost model. Furthermore, we also intend to investigate strategies for reorganizing the network in order to keep the system globally effective even if local evolutions due to network and/or subscription changes lead to a deterioration of global system performance.

Data stream clustering as described above also contributes to fulfilling the third goal of effective multi-query optimization. In every subnet, the speaker-peer analyzes the registered subscriptions and identifies common subexpressions. These common subexpressions are evaluated once in this subnet by executing a subscription rule corresponding to a common subexpression at a suitable peer. Rather than individually evaluating this subexpression in each of the original subscriptions, the subscriptions are rewritten to utilize the newly generated and specialized data stream stemming from the common subexpression. Besides reducing the workload of the affected peers, network traffic might be further reduced. For instance, a common task will be aggregating sensor data. Instead of transmitting the whole dataset to every peer performing the same aggregation, it will be executed near the data source and only the aggregated results, which will constitute a smaller data volume, will be delivered to the respective peers. Furthermore, existing aggregated data streams in the system can be reused to compute more common aggregates similar to the roll up and the cube operations in data warehousing.

Figure 4 shows the query evaluation strategy using the example scenario from Section 1. The symbols at the network connections represent groups of elements. The diamond represents the elements `bt`, `pr`, and `et`, the circle represents `os`, the triangle represents `CO2` and `SO2`, and the rectangle represents `id`, `time`, `x`, and `y`. Projections cause symbols to disappear as their corresponding elements are filtered out of the stream. Selections remove certain instances of elements that do not fulfill the selection predicates which is depicted as dotted symbols. An exclamation mark denotes a change in data representation, e.g., the introduction of the *alert* element at SP_2 in the result for the query at P_0. In our example, the introduction of the *gas* element in the answer for the query at P_2 is supposed to take place at P_2 itself and therefore does not show up in the network. The decision whether to perform the FluX subscription evaluation at P_2, SP_1, or SP_2 is made by the optimizer and is based on factors like computational power and current load factor of peers.

The sample query evaluation plan in Figure 4 depicts the situation after the data stream and the two queries of Section 1 have been registered in the network of Figure 1. Furthermore, the query optimizer has already optimized the queries and integrated them into the system. First, the elements `bt`, `pr`, and `et` are removed from the stream by a projection operator. To reduce network traffic, the optimizer chooses to install the mobile code of the appropriate projection operator as close to the data source as possible. Since the data source P_4 is a simple sensor without query processing capabilities and is therefore not able to perform the projection by itself, the projection operator has to be installed and executed in the network at super-peer SP_3. The resulting data stream is routed only once (as one data stream cluster) to SP_2, although it is needed twice in the system. Therefore, the optimizer decides to replicate the data stream at SP_2 to obtain two identical versions of the stream. The decision of how to route and where to replicate the stream is simply made by pursuing the goal of minimizing the number of hops each stream has to go from its source to its recipient in the network. Of course, more sophisticated optimization goals and routing strategies can be employed here. We will examine this in future work. At

SP_2, the stream with destination P_2, which is the fire department, is again reduced by a projection operator removing element os. The remaining stream is forwarded to P_2 via the shortest path, in this case over SP_1. The rest of the query evaluation, consisting of the introduction of the *gas* element, is performed at P_2 itself. The stream with destination P_0 is also filtered at SP_2, this time using a projection and a selection as demanded by the respective query. Also, the new *alert* element in the query result is already introduced at SP_2. The resulting stream is then forwarded to P_0, again using the shortest path which is via SP_0. In general, the shortest path is not unique and depends on the underlying network topology. In the case of multiple shortest paths, one appropriate path among them is chosen.

Continuing our example from Section 1, we now take a look at a more complicated situation. Let us assume that peer P_1 represents a collection of weather sensors delivering a virtual data stream registered at super-peer SP_0. Each sensor reading contains the identifier of the corresponding sensor (`id`), a timestamp (`time`), the GPS coordinates of the sensor (`x`, `y`), and measurements for wind (`wind`), temperature (`temp`), humidity (`hum`), and air pressure (`ap`). Sensor readings for wind consist of wind strength (`strength`) and wind direction (`direction`). The resulting data stream corresponds to the following DTD.

```
<!ELEMENT reading (id, time, x, y,
                   wind, temp, hum, ap)>
<!ELEMENT id (#PCDATA)>
...
<!ELEMENT wind (strength, direction)>
<!ELEMENT strength (#PCDATA)>
<!ELEMENT direction (#PCDATA)>
...
```

We now further assume that the fire department at P_2 registers a new query at SP_1 in addition to the one already registered in Section 1. This new query requires the data from P_4 to be joined with data from P_1. The fire department is interested in finding out how strong and from which direction the wind blew at the point in time and at the place a gas concentration was measured. Therefore, it joins the data of the gas sensors from P_4 with that of the weather sensors from P_1. The join tries to find for each measured gas concentration a sensor reading for wind strength and direction that was close to the gas measurement in terms of both, the point in time the respective sensor readings where created and the geographical location at which the corresponding sensors where located. This can be achieved by using the bestmatch join operator [17].

One possibility to compute the join would be to filter P_1's data stream accordingly at SP_0 and route the resulting stream directly to SP_1, where the join processing takes place and the result gets delivered to P_2. This would probably be the best solution if no data from peer P_1 is needed anywhere else in the network. However, when P_3 also requests data from P_1, it might be better to route a data stream with the data for both P_2 and P_3 from SP_0 to SP_2 first and then split the stream at SP_2, routing P_3's part directly to P_3. The remaining stream for peer P_2 could then be routed to SP_1, where the join processing could take place. But if the join is known to produce a relatively small result compared to the input streams, it would probably be better in terms of network traffic to process the join already at SP_2 and then route the result to P_2 via SP_1. This is an example of a more difficult decision that has to be made by the StreamGlobe query optimizer.

4.2 Query Processing

Let us now outline some basic concepts used for in-network query processing. Query execution in StreamGlobe focuses on processing streaming data and therefore employs *push-based* evaluation strategies—in contrast to traditional query engines where data is normally "pulled" from subordinate operators, e.g., by using the iterator model.

First, we will explain how filtering operators are executed. As outlined before, filtering operators perform a projection of a data stream on the required parts of the entire schema and a selection according to predicates of a subscription rule. Since the basic schema of the original data stream remains the same[1] (besides discarding unnecessary information), projection can be done on-the-fly without the need of buffering parts of the data stream. Performing selections is somewhat more difficult, because in the worst case data cannot be propagated before the predicate is evaluated, which renders buffering inevitable. Thus, we restrict filtering operators to only employ predicates referring to a single data object of the data stream. Therewith, at most the current data object has to be buffered for being able to propagate the filtered data stream. Hence, we can implement these operators scalably and efficiently using automata-based techniques as described in [18] or the new FluX query engine which was developed in cooperation with our group and will be sketched in the remainder of this section.

In order to evaluate subscription rules on data streams, we employ novel optimization techniques, called *FluX* [19], for minimizing memory buffer consumption during the execution of XQueries on streaming data. FluX is an intermediate language extending the XQuery syntax by event-based processing instructions which enables conscious handling of main memory buffers. The key idea of the FluX query language is the novel *process-stream* statement `{ ps` $\$x$`:` ζ `}` for event-based (streaming) processing of a substream assigned to a variable $\$x$. It processes the data stream by means of a list of *event-handlers* ζ. Each event handler is of one of the two forms

- `on` a `as` $\$y$ `return` α
- `on-first past(`S`) return` α

with α being an arbitrary subexpression, a being the label of a tag, and S being a set of labels of XML tags. An "`on` a" handler is executed if an opening tag labeled a is encountered in the stream of $\$x$. The subsequent elements

[1]In particular, the order of elements is preserved.

of the data stream are labeled as a substream $\$y$ and used to evaluate the subexpression α (which may in turn be a process-stream statement or traditional XQuery). The latter "`on-first`" handler is executed if no more elements labeled s with $s \in S$ will be encountered in the stream being currently processed and triggers the evaluation of α. In general, an arbitrary query cannot be evaluated purely on-the-fly without buffering, e.g., if the sequence of elements in the query is different from that in the input data stream. Hence, a FluX query consists of a purely streaming part using our novel syntax and of embedded traditional XQuery, which is evaluated on previously buffered parts of the data stream. The main challenge is to rewrite an XQuery into a corresponding FluX query which evaluates this query using as many of the event-based methods as possible and thereby minimizing buffer usage. In [19], an algorithm which utilizes order constraints on the elements imposed by the DTD of the data stream is presented to achieve this goal.

Rewriting XQuery into FluX is based on generating a *safe* FluX query. That is, an XQuery subexpression of a FluX query operating on buffered data must only reference—e.g., by path expressions or other variables—parts of the data stream which will not be encountered any more after this expression has been evaluated. Thus, the query engine can easily populate buffers with the needed parts of the data stream and provide these buffers for the execution of the buffer-based parts of the FluX query. The second query of our example scenario using the given DTD would be rewritten into FluX as follows.

```
{ps stream("firefighters")
  on reading as $m return
    {ps $m:
      on-first past() return <gas>;
      on id as $id return {$id};
      on x as $x return {$x};
      on y as $y return {$y};
      on CO2 as $CO2 return {$CO2};
      on SO2 as $SO2 return {$SO2};
      on-first past(*) return </gas>; } }
```

This FluX query is purely event-based (outputting the values of the substreams in the "`on`" handlers can be done on-the-fly) and hence needs no buffering at all. "`on-first past(*)`" is a shortcut for the set S containing all possible labels in this substream and is therefore executed after all other elements have been written. More details on FluX together with an experimental evaluation can be found in [19].

Summarizing, FluX enables query evaluation on data streams with very low memory consumption and thus provides for a scalable evaluation of subscription rules. However, some subscription rules might possibly need unbounded buffering, e.g., subscriptions containing joins or special aggregates. In such cases, unbounded buffering is precluded by requiring users to specify window constraints. These allow for a scalable execution on infinite data streams.

5 Implementation Status

As of the writing of this paper, we have implemented the basic infrastructure of StreamGlobe, building on the Globus Toolkit, and we are able to establish an overlay P2P network between peers. We have also completed a prototype implementation of the FluX streaming query engine for evaluating subscription rules. This query engine is currently being integrated into the StreamGlobe system. At the moment, the optimization techniques of Section 4 are developed and implemented. A first prototype system of StreamGlobe including all the basic features presented in this paper will be operational by the end of the year.

6 Conclusion and Future Work

In this paper, we have described the ongoing development of our StreamGlobe system. StreamGlobe is focused on meeting the challenges that arise in processing data streams in an ad-hoc P2P network scenario. It differs from other data stream systems in not only efficiently locating and querying data streams, but also optimizing the data flow in the network using expressive in-network query processing techniques. This is basically achieved by pushing operators for query processing into the network. Continuous reoptimization leads to an adaptive and self-optimizing system which enables users to carry out powerful information processing and retrieval. StreamGlobe builds on and extends the Globus Toolkit, a reference implementation of the Open Grid Services Architecture (OGSA) for grid computing, and serves as a research platform for our future work.

Future research will cover further topics in query processing on streaming data, optimization methods for distributed data stream processing, load balancing and quality-of-service aspects [4] in a distributed data stream management system. In detail, this will include augmenting the FluX query engine to support windowing operators like aggregations and joins. It will also comprise improving the optimization component by taking into account reorganisation issues to keep the system effective as well as synchronization aspects, e.g. for distributed join processing on various streaming inputs. Furthermore, we will continue to examine routing approaches for our hierarchical network organisation and conduct advanced research concerning the combination of multiple query processing operators, predicate comparisons in the context of query clustering, and the minimization of memory requirements during query evaluation. Eventually, support for content-based query subscriptions will be added to StreamGlobe.

Acknowledgments. Franz Häuslschmid and Angelika Reiser provided helpful comments on earlier revisions of this paper. Christoph Koch, Stefanie Scherzinger, and Nicole Schweikardt realized together with one of the authors the FluX [19] query engine.

References

[1] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: a self-

organizing structured P2P system. *ACM SIGMOD Record*, 32(3):29–33, Sept. 2003.

[2] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, R. Motwani, I. Nishizawa, U. Srivastava, D. Thomas, R. Varma, and J. Widom. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin*, 26(1):19–26, Mar. 2003.

[3] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. *The VLDB Journal*, 10(1):48–71, Aug. 2001.

[4] R. Braumandl, A. Kemper, and D. Kossmann. Quality of Service in an Information Economy. *ACM Transactions on Internet Technology*, 3(4):291–333, Nov. 2003.

[5] I. Brunkhorst, H. Dhraief, A. Kemper, W. Nejdl, and C. Wiesner. Distributed Queries and Query Optimization in Schema-Based P2P-Systems. In *Proc. of the Intl. Workshop On Databases, Information Systems and Peer-to-Peer Computing*, Berlin, Germany, Sept. 2003.

[6] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proc. of the Intl. Conf. on Very Large Data Bases*, pages 215–226, Hong Kong, China, Aug. 2002.

[7] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proc. of the Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, Jan. 2003.

[8] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 379–390, Dallas, TX, USA, May 2000.

[9] L. Chen, K. Reddy, and G. Agrawal. GATES: A Grid-Based Middleware for Processing Distributed Data Streams. In *Proc. of the IEEE Intl. Symp. on High-Performance Distributed Computing*, Honolulu, HI, USA, June 2004. To appear.

[10] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik. Scalable Distributed Stream Processing. In *Proc. of the Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, Jan. 2003.

[11] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. J. Carey, A. Sundararajan, and G. Agrawal. The BEA/XQRL Streaming XQuery Processor. In *Proc. of the Intl. Conf. on Very Large Data Bases*, pages 997–1008, Berlin, Germany, Sept. 2003.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, June 2002. http://www.globus.org/research/papers/ogsa.pdf.

[13] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The Intl. Journal of Supercomputer Applications and High Performance Computing*, 15(3):200–222, Aug. 2001.

[14] The Globus Alliance. http://www.globus.org.

[15] Q. Huang, C. Lu, and G.-C. Roman. Spatiotemporal Multicast in Sensor Networks. In *Proc. of the Intl. Conf. on Embedded Networked Sensor Systems*, pages 205–217, Los Angeles, CA, USA, Nov. 2003.

[16] M. Keidl, A. Kreutz, A. Kemper, and D. Kossmann. A Publish & Subscribe Architecture for Distributed Metadata Management. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, pages 309–320, San José, CA, USA, Feb. 2002.

[17] A. Kemper and B. Stegmaier. Evaluating Bestmatch-Joins on Streaming Data. Technical Report MIP-0204, Universität Passau, 2002.

[18] C. Koch and S. Scherzinger. Attribute Grammars for Scalable Query Processing on XML Streams. In *Proc. of the Intl. Workshop on Database Programming Languages*, pages 233–256, Potsdam, Germany, Sept. 2003.

[19] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier. Schema-based Scheduling of Event Processors and Buffer Minimization on Structured Data Streams. In *Proc. of the Intl. Conf. on Very Large Data Bases*, Toronto, Canada, Aug. 2004. To appear.

[20] J. Krämer and B. Seeger. PIPES - A Public Infrastructure for Processing and Exploring Streams. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 925–926, Paris, France, June 2004.

[21] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering Queries Using Views. In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 95–104, San José, CA, USA, May 1995.

[22] S. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously Adaptive Continuous Queries over Streams. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 49–60, Madison, WI, USA, June 2002.

[23] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamritham. Materialized View Selection and Maintenance Using Multi-Query Optimization. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 307–318, Santa Barbara, CA, USA, May 2001.

[24] OGSA-DAI. http://www.ogsadai.org.uk.

[25] M. T. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP – Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks. In *Proc. of the Intl. Workshop on Agents and Peer-to-Peer Computing*, pages 112–124, Bologna, Italy, July 2002.

[26] T. K. Sellis. Multiple-Query Optimization. *ACM Trans. on Database Systems*, 13(1):23–52, Mar. 1988.

[27] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer Data Management Systems. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 539–550, Paris, France, June 2004.

[28] B. Yang and H. Garcia-Molina. Designing a Super-Peer Network. In *Proc. of the IEEE Intl. Conf. on Data Engineering*, pages 49–60, Bangalore, India, Mar. 2003.

[29] Y. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proc. of the Intl. Conf. on Very Large Data Bases*, pages 136–145, Athens, Greece, Aug. 1997.

[30] Y. Yao and J. Gehrke. The Cougar Approach to In-Network Query Processing in Sensor Networks. *ACM SIGMOD Record*, 31(3):9–18, Sept. 2002.

[31] Y. Yao and J. Gehrke. Query Processing for Sensor Networks. In *Proc. of the Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, Jan. 2003.