Processing Publish/Subscribe Queries over Distributed Data Streams[*]

Oana Jurca, Sebastian Michel,
Alexandre Herrmann, Karl Aberer
Ecole Polytechnique Fédérale de Lausanne, Switzerland
firstname.lastname@epfl.ch

ABSTRACT

We address the problem of processing continuous multi-join queries, over distributed data streams, making use of existing work in the field of publish/subscribe systems. We show how these principles can be ported to data streams, by enriching the common query model with location dependent attributes. Users can subscribe to a set of sensor attributes, a service that requires processing multi-join correlation queries. The goal is to decrease the overall network traffic consumption by removing redundant subscriptions and eliminating unrequested events close to the publishing sensors. This is non-trivial, especially in the presence of multi-join queries without any central control mechanism. Our approach is based on the concept of filter-split-forward phases for efficient subscription filtering and placement inside the network. We report on a performance evaluation using a real-world dataset, showing the suitability of our approach to reduce the overall data traffic.

1. INTRODUCTION

The advances of sensing technologies, in particular wireless sensors, allow for continuous and detailed monitoring of the real world. Application scenarios are manifold, ranging from traffic or health monitoring, to observing phenomena in environmental sciences. In the latter, scientists use a wide variety of observation networks, ranging from small deployments, to interconnected sensor stations spanning hundreds of square kilometers, with different types of sensors attached. Experiments range in scale from on-site to arbitrarily large and complex, consisting of heterogeneous sensor types, often operated by different organizations.

This allows for sophisticated queries over the existing data, resulting in complex event descriptions involving multiple sensors. Sensors can be addressed explicitly, but it is more likely users are interested in one or more data types within a particular spatial region. Queries in this context are so-called continuous queries, or subscriptions, and return results to the subscriber whenever a particular condition (event) is satisfied. The main difficulty is to place subscriptions efficiently inside the network, such that sensor readings (several orders of magnitude more numerous) are filtered close to their source. This becomes challenging in the case of subscriptions over multiple data sources, because they require "joins" over those sources. For large numbers of subscriptions, they will naturally overlap, and so will their result sets, hence, wasting network resources which are often seen as the limiting factor for achieving acceptable response times.

A typical example of a subscription is looking for a combination of high solar radiation, sufficient snow height, and low wind speed. Analogously, high temperature, small air humidity, and medium water temperature, could indicate a pleasant day at the lake.

We present a scalable approach that can be applied over any sensor network middleware, with the following contributions: We tailor state of the art publish/subscribe algorithms for sensor networks, extending the subscription language, and adapting subscription filtering and event matching. We propose a novel algorithm for placing user subscriptions in the system, composed of filtering, splitting and forwarding phases: this injects fewer subscriptions, which coupled with our event propagation algorithm avoids result overlap and lowers network traffic. We report on a performance evaluation using a real-world dataset, showing the reduction in overall data traffic.

2. RELATED WORK

So far, only few architectures support more complex processing on interconnected sensor networks. Sgroi et al. [11] suggest basic abstractions, a standard set of services, and an API to free application developers from the details of the underlying sensor networks, focusing on systematic definition and classification of abstractions and services. Both IrisNet [6] and Hourglass [12] propose an Internet-based infrastructure for connecting sensor networks to applications. [5] and [10] provide hierarchical data stream query processing to acquire, filter, and aggregate data from multiple devices in a static environment, with the focus on filtering the useful data as close as possible from the producing device.

Work in the field of operator placement [15, 1, 9] is similar to ours. However, they require some form of global knowledge and the query rewriting techniques do not achieve a high query load reduction, while the result sets still overlap; thus both queries and, more importantly, result tuples are redundantly pushed in the system.

Publish/subscribe has been recognized as an efficient com-

[*]The work presented in this paper was partially supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

munication paradigm for sensor networks: [13] employs topic-based publish/subscribe and aggregates data over one topic. [7] aims at reducing the communication traffic by using a reduced number of paths inside the system, identified by an augmented distance vector protocol. A semi-probabilistic approach, [4], combines deterministic (due to a match) with probabilistic forwarding of publications (when there is no match) and trades reliability, for publication traffic. More recently, [3] achieves less duplication, but assumes a centralized processing location, and introduces false positives.

3. MODEL

We model sensors as data sources (producers) and users as data consumers (subscribers). A sensor data type is modeled as a data attribute and together with its location constitutes its data source advertisement (DSA). Each sensor reading is a publication, characterized by its data type, value, location and publication time. A user subscription is a set of filters expressing ranges either over explicitly named sensors (identified subscription), or over types of data streams, bounded to a specified region (abstract subscription).

Typically, users are interested in correlated data from multiple data sources, but publication timestamps are seldom identical over different sensors. Therefore, the *temporal correlation distance* δ_t is the maximum acceptable difference between publication timestamps to be considered as correlated. Similarly, for data streams from a region, the *spatial correlation distance* δ_l gives the maximum acceptable difference between different sensor locations such that their publications are considered to be correlated. If event correlation is independent of spatial proximity we set $\delta_l = \infty$. Most applications set δ_t and δ_l as constants, because correlations have the same meaning through out the application.

A publication matches a subscription if the corresponding filter evaluates to true. A subscriptions's result set contains complex events, which are sets of publications fulfilling the required correlation distance(s), but also a completeness constraint: each filter, over an attribute or sensor, must have a matching publication.

To optimize the processing of event and subscription propagation, we will rely on *subscription subsumption*: A subscription s is subsumed by a set of subscriptions, if any complex event matching s, matches at least a subscription in the set.

Our system consists of sensor nodes, representing sources of data streams, and relaying nodes, all connected in an acyclic graph. Data propagation in our network happens three-fold, for each group of data: advertisements, subscriptions, and events, with subscriptions following the reverse dissemination path of advertisements and events following the reverse dissemination path of subscriptions.

4. ALGORITHMS

We exemplify the data storage in Figure 1, for a node n, with five neighbors: two sensor nodes (nodes 1 and 2), and three nodes (3, 4 and 5) hosting users with subscriptions. For simplicity, we do not show data relayed by n to its neighbors. At n, advertisements received from each neighbor m or local sensors, are stored in separate structures, identified with DSA^m ($DSA^1 = \{DSA_1\}$, $DSA^2 = \{DSA_2\}$) and DSA^{local}, respectively (empty, not shown). Similarly, received subscriptions are stored in structures identified with S^m ($S^3 = \{s_3\}$, $S^4 = \{s_4\}$ and $S^5 = \{s_5\}$) and S^{local} (empty, not shown). The received events (forwarded by neighbors or published by local sensors) are stored together, in a structure identified with U.

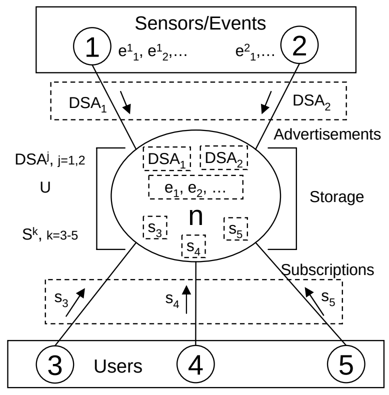

Figure 1: Data structures at node n

4.1 Advertisement Propagation

Advertisements are flooded in the network, as the number of data sources is expected to be far less numerous than the number of subscriptions, themselves orders of magnitude less frequent than publications. At a node, all received advertisements (from neighbors and local data sources) are propagated to all neighbors (except the originating one), and stored in DSA structures, per neighboring node, plus local data sources. This ensures that any incoming subscription can immediately follow the matching advertisements' reverse dissemination path.

4.2 Subscription Propagation

Nodes receive subscriptions from local users and neighboring nodes, and should forward them toward corresponding sensors, following the reverse path of matching sensor advertisements. A subscription matches events only if all its attributes have corresponding sensors. To reduce storage needs and subscription traffic, nodes do not forward such subscriptions and subsumed subscriptions.

Set Filtering

The most efficient constraint query filtering technique, known in general as **set coverage** or **set subsumption**, was proven to be co-NP complete [14]. In [8], we propose a probabilistic algorithm for publish/subscribe systems, which guarantees detection of set subsumption with a configurable probability of error given as input. We call this algorithm **set filtering**, and find it appropriate for distributed data streams, like sensor networks, where data sources could be unreliable, user subscriptions very numerous and transmitted data can be lost due to traffic congestion and link failure ([2, 7]).

However, **set filtering** as defined in the context of publish/subscribe systems cannot be directly applied to distributed sensor networks, because all subscriptions and events are defined over the complete set of attributes; in fact, all published data in such systems is already correlated. To solve **set filtering** for distributed data streams, we define subsumption within our model, which also considers the location and timestamp meta-attributes, aside from the non-correlated data attributes. The timestamp attribute is orthogonal to the subsumption process, because it affects data content (the streams) and its correlation, not user subscriptions (as long as δ_t is constant). The location meta-attribute has a direct impact, even more so as we support identified and abstract subscriptions. For identified subscriptions, each sensor in the system acts as one attribute, while for abstract subscriptions, data types act as data attributes. The location meta-attribute adds another dimension to the problem, and due to the location domain's containment property, we can treat it as another data attribute (e.g., a location region can be contained in another region, or union of such regions). After this translation we still have the problem of non-correlated data: subscriptions are defined over different attribute subsets. We cannot assume that a subscription with a missing attribute requests the whole corresponding value space (instead, values for such an attribute are unrequested and should not be forwarded to the end user).

Divide and Conquer

We have opted for a "divide-and-conquer" solution, taking into account that subscriptions with more attributes are more restrictive than those with less attributes and demand more processing for data correlation. On the other hand, the former's matching data could also match the latter. Our strategy is, for each new subscription, to first check subsumption against subscriptions over the same attribute set, and in case of a negative answer, to divide it into simpler subscriptions, which can be compared with subscriptions over subsets of the initial attribute set. We need a method to divide the original subscription into subsets of attributes (and the filters over them) arises. We check data source advertisements to drop subscriptions without sources and to guide answerable subscriptions towards the matching sensors. We need for each sensor only the filter, part of the original subscription, that affects it. In fact, advertisement reverse path gives us a deterministic choice for subscription splitting, and we implicitly postpone the subsumption processing to the node where the paths diverge. If all subscription subparts are subsumed along the paths, this process can actually detect subsumption against subscriptions over different attribute subsets.

Subscription Placement

Subscription forwarding makes sense only if the subscription can be answered by the existing data sources. A subscription might be forwarded as a whole, or as subsets of filters (split): one subset for each neighbor with matching advertisements, over the attributes common to those advertisements and the processed subscription. As a consequence of this split and forward process, nodes forward subscriptions either as the complete set of filters given by a user, or as filters subsets. We refer to a (sub)set of filters as a correlation operator, because it requires data over different streams and must correlate them on time (and maybe also location). When such an operator is addressing a single attribute, we call it a simple operator, and it does not suffer further splitting on its way to the matching sensor(s), though it might be filtered out along the way.

The filtering is done on the set of incoming subscriptions, whether they are from local users or a neighbor, against the uncovered subscriptions, because the covered subscriptions are redundant to the filtering problem. Both covered and uncovered subscriptions must to be stored (placed): even though only uncovered subscriptions are candidates for forwarding to neighbors, all subscriptions define the correlations needs of the neighbors or local users. User subscriptions that were not filtered out might require some other information than existing subscriptions and must be forwarded (and placed) in the system towards the matching data sources. If these data sources have different paths, the subscription is correspondingly divided, and subsequent filtering and splitting will eventually determine whether it is subsumed (and thus no new filters arrive to the data sources).

4.3 Event Propagation

A sensor node generates (simple) events for each sensor reading. Because events can be very numerous, for efficiency reasons they should be forwarded in the system only if interested users exist: a subscription or correlation operator addressing the corresponding attribute is present at the node. On the path from the publishing node to the matching users, an event is forwarded (or on the contrary, dropped) only if it is part of a complex event matching a correlation operator, or if it matches a simple operator.

At each node, all received simple events are stored and indexed by their timestamps, to facilitate time correlation. A potential complex event, containing the incoming simple event e, can only be present in a δ_t length sliding window over the received events set. Each set of simple events, in an instance of the sliding window, must be checked against all local subscriptions, but only against the set of uncovered subscriptions from neighbors. For each subscription, the set of simple events is checked against the subscription filters, and if at least one event per subscription attribute is still left, we know a matching complex event exists. The simple events from the matching complex events are sent to the neighbor having forwarded the subscription, if they have not been previously forwarded to that neighbor, in accordance to the publish/subscribe paradigm.

5. EXPERIMENTS

We study the performance of our algorithm while comparing it against the baseline of a **naive approach**. This approach emphasizes the problem of traffic load in multi-join query processing, as it forwards all received queries (no filtering) and constructs result sets per query (no optimization for result set overlap). Our novel approach, **Filter-Split-Forward**, described in Section 4, detects correlation operators subsumption, through efficient filtering, system state driven splitting and placement, all based strictly on local interaction and further reduces traffic by a publish/subscribe forwarding of events.

Our Java (JSE5.0) implementation has been evaluated in a distributed environment, on a cluster of quadcore 2.4GHz machines running Linux (Debian Etch). Each node in our system runs on a virtual machine, simulating a less powerful device, with 256MB RAM, and 1.5GB HDD. The virtual nodes were created with paravirtualization (Xen No HW emulation) at a ratio of 30 nodes per quadcore.

We run the experiments on a real data set from the Sensorscope project (http://sensorscope.epfl.ch). We have selected 5 measurement data types (Ambient Temperature, Surface Temperature, Relative Humidity, Wind Speed, and Wind Direction) as attributes in our setting, and 10 base stations as data sources. Subscriptions are generated by creating ranges over the sensor streams, centered around the median values in the corresponding stream, with an offset drawn from a Pareto distribution with a skew factor of 1. We target all locations with the same number of subscriptions, which we vary across each experiment.

We tested our approach in a smaller network of 60 nodes and larger network of 100 nodes, out of which 50 have a sensor attached, evenly distributed over 5 attribute types. As such we simulate 10 different locations or subnetworks. In the smaller scale experiment, we vary the number of user subscriptions from 100 to 1000, and we measure the performance after every new batch of 100 subscriptions. The subscription attribute set varies between 3 to 5 attributes chosen randomly, to ensure a good mixture of attribute subsets. This generates less subscription overlap. In the larger

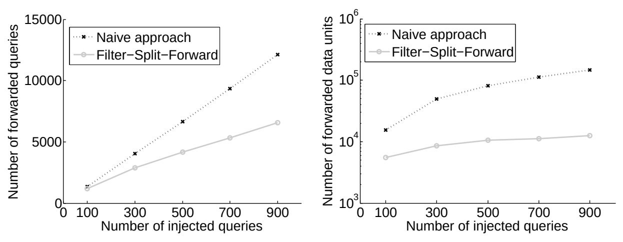

Figure 2: Subscription and event load

scale experiment, subscriptions have 5 attributes each, we vary their number from 100 to 900 and we report the traffic load after each 100 subscriptions batch.

The first metric, **subscription load**, reflects the entire load of subscriptions forwarded in the network, and increases every time a correlation operator is forwarded to a neighboring node. It is influenced by the initial number of user subscriptions having matching data sources, by node connectivity and distribution, and also by the subscription propagation policy. The results are similar for the two settings, so we depict only the larger scale results. Overall subscription overlap increases with the number of user subscriptions. Our approach reduces the subscription set (Figure 2, left) due to group subsumption, associated with the splitting phase of the algorithm. The performance is slightly worse for the smaller scale experiment because of subscriptions having different attribute sets, thus less subsumption. However, the splitting phase allows the comparison between subscriptions over different (but overlapping) attribute sets, by comparing the simpler split operators and achieving reduction results similar to cases where all subscriptions are over the same attributes.

The key issue of complex events traffic is comprised in the **publication load** metric, which quantifies the result sets forwarded over each network link. This metric is influenced by the number of operators residing at each node, because each one generates a result set, but also by the efficiency of event forwarding. It also depends on node connectivity and distribution. The Filter-Split-Forward good performance in the larger scale experiment (Figure 2, right) is due to smaller number of forwarded operators and sharing of dissemination costs between overlapping operators. The gain is larger than for the smaller scale experiment because the decrease in result sets is amplified by the forwarding over more links, as the network is larger.

Part of our proposed algorithm is probabilistic, which can result in lost information. This happens when user subscriptions are falsely assumed to be subsumed, as a result of the probabilistic process. Events matching such subscriptions will not arrive to the user, unless they also match other overlapping subscriptions. The **event recall** metric reflects how many useful events are lost by our algorithm as compared with the naive approach which is deterministic. The measured accuracy can reach 100%, and is on average 99% for the large scale experiment. However, for the small scale experiment, the recall is around 93%, which we think is also acceptable in most application scenarios. To achieve a higher recall, one can think of adapting the probabilistic filtering to be more conservative. In addition, in scenarios that require not to lose any event, the subscriptions can be made coarser, i.e., the specified filter ranges could be enlarged, requiring a final local filtering step. However, this would increase the number of forwarded events, and the overall traffic. Reducing either the traffic, either the number of missed events creates a tradeoff, upon which the user has to decide.

6. CONCLUSION AND OUTLOOK

We have presented a general framework for processing publish/subscribe queries over distributed sensor networks. The requirements imposed on the underlying sensing infrastructure are almost negligible, hence, the integration of a large number of heterogeneous sensor networks under one common light-weight event processing model becomes possible. In our approach, multi-join queries are passed along communication links, from one node to other nodes, towards the data sources. We employ so called filter-split-forward phases for efficient subscription filtering and placement inside the network. We have implemented our approach and have conducted a performance evaluation showing its suitability for reducing the traffic load. As future work, we will have a look at ranking batches of events, for more efficient event propagation, focusing only on the top-ranked items. This is in particular interesting for subscription queries posed by users with large numbers of matching events.

7. REFERENCES

[1] Y. Ahmad et al. Network-aware query processing for stream-based applications. *VLDB*, 2004.

[2] I. F. Akyildiz et al. A survey on sensor networks. *IEEE Communications Magazine*, 40(4), 2002.

[3] B. Chandramouli and J. Yang. End-to-end support for joins in large-scale publish/subscribe systems. *VLDB*, 2008.

[4] P. Costa et al. Publish-subscribe on sensor networks: A semi-probabilistic approach. *MASS*, 2005.

[5] M. Franklin et al. Design Considerations for High Fan-in Systems: The HiFi Approach. *CIDR*, 2005.

[6] P. B. Gibbons et al. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4), 2003.

[7] C. Hall et al. A content-based networking protocol for sensor networks. Technical Report CU-CS-979-04, Department of Computer Science, University of Colorado, Aug. 2004.

[8] A. M. Ouksel et al. Efficient Probabilistic Subsumption Checking for Content-Based Publish/Subscribe Systems. *Middleware*, 2006.

[9] P. Pietzuch et al. Network-aware operator placement for stream-processing systems. *ICDE*, 2006.

[10] S. Rooney et al. Techniques for Integrating Sensors into the Enterprise Network. *IEEE eTransactions on Network and Service Management*, 2(1), 2006.

[11] M. Sgroi et al. A service-based universal application interface for ad hoc wireless sensor and actuator networks. *Ambient Intelligence*, 2005.

[12] J. Shneidman et al. Hourglass: An Infrastructure for Connecting Sensor Networks and Applications. Technical Report TR-21-04, Harvard University, EECS, 2004.

[13] E. Souto et al. Mires: a publish/subscribe middleware for sensor networks. *Personal Ubiquitous Comput.*, 10(1), 2005.

[14] D. Srivastava. Subsumption and indexing in constraint query languages with linear arithmetic constraints. *Annals of Mathematics and Artificial Intelligence*, 8(3–4), 1992.

[15] U. Srivastava et al. Operator placement for in-network stream query processing. *PODS*, 2005.