ProSem: Scalable Wide-Area Publish/Subscribe

Badrish Chandramouli * Jun Yang
Pankaj K. Agarwal Albert Yu Ying Zheng
Department of Computer Science
Duke University, Durham, NC 27708, USA

ABSTRACT

We demonstrate ProSem, a scalable wide-area publish/subscribe system that supports complex, stateful subscriptions as well as simple ones. One unique feature of ProSem is its cost-based joint optimization of both subscription processing and notification dissemination. ProSem uses novel *reformulation* techniques to expose new alternatives for processing and disseminating data using standard stateless content-driven network components.

Categories and Subject Descriptors: C.2.4 [Computer Communication Networks]: Distributed Systems—Distributed applications

General Terms: Design, Experimentation, Performance

1 Introduction

The ubiquity of the Internet has generated unprecedented demand for information. A publish/subscribe system is well-suited to the task of matching data recipients (*subscribers*) to data providers (*publishers*). There has been a lot of work on publish/subscribe systems in both the database and networking communities. The initial systems were topic-based (e.g., *SCRIBE* [3]), whereas the next generation of systems (e.g., *Gryphon* [1]) supported more complex subscriptions that could specify predicates over the contents of individual *events*. However, neither model captures the complex data needs of modern applications. For instance, users often want data that are correlated or aggregated over event history and database of related information. Moreover, users may prefer to receive updates only when certain user-defined *notification conditions* are met. Some efforts have been made by the database community in supporting complex subscriptions (e.g., *Cayuga* [6]). These systems focus on subscription processing at a server, and do not address how to notify subscribers over a network. Standard practices—unicasting notifications or letting subscribers poll for notifications—have severe scalability problems with a large number of subscribers over a wide area.

Fundamentally, a wide-area publish/subscribe system must handle subscription processing as well as notification dissemination (henceforth called *prosemination* to emphasize their inseparability). We have developed holistic prosemination techniques for many types of subscriptions. These techniques offer orders-of-magnitude improvement in performance over traditional, non-holistic schemes. Hence, we are building a publish/subscribe system called *ProSem*, named after "prosemination" to signify our novel holistic approach to prosemination. Besides joint optimization, ProSem has several additional features that also distinguish it from other systems:

- ProSem supports complex, stateful subscriptions, e.g.: "keep me informed of stocks with the minimum price-to-earning ratio in a certain risk range," "notify me when the price of IBM has changed by more than 3%," etc. Furthermore, ProSem aims at supporting such subscriptions on an Internet-scale. Beyond exploiting common subscription predicates, ProSem uses subscription indexing techniques specially developed for certain subscription types to scale to millions of subscriptions.
- Using a novel technique called *reformulation*, ProSem is able to support complex, stateful subscriptions on top of standard network substrates that otherwise only support stateless subscriptions. This approach avoids pushing complicated application logic into the network, enforces a clean interface between the server and the network, and allows the use of an "off-the-shelf" *content-driven network (CN)* [5]—a term that encompasses many network substrates including *content-based networks* [2] and overlay networks supporting range searches.
- ProSem recognizes that the best prosemination strategy may differ for different events and different groups of subscriptions. Therefore, ProSem uses cost-based optimization to choose the best prosemination strategy among a set of available alternatives at run-time, according to user-defined system-level objectives.

Although some building blocks of ProSem have been introduced in recent work [4, 5], ProSem is the first attempt at combining and implementing these building blocks across various subscription types in one system. ProSem's run-time optimization of prosemination is also original.

2 Preliminaries

We model a publish/subscribe system as consisting of (1) *publishers* who publish *events* which are modeled as updates to a database, (2) *subscribers* who declare their interests in data (called *subscriptions*) as queries over the database, and (3) the middleware—a *server* and an optional network of *brokers*—responsible for supporting the subscriptions. As a running example, consider the database view `Stock(Symbol, Price, Risk, PER)` tracking the up-to-date information for each stock, including the stock symbol, current price, risk factor, and price-to-earning ratio, a popular measure of stock quality. For simplicity, suppose that published events follow the schema $\langle$`Symbol, Price, Risk, PER`$\rangle$, and the attributes reflect their new values after the update.

A subscription is *stateless* if we can determine whether and how to update it by examining an incoming event itself, without referring to the database or event history. For instance, a subscription of the form "`SELECT * FROM Stock WHERE` $x_1 \leq$ `Risk AND Risk` $\leq x_2$" is stateless; it is affected by any event with `Risk` $\in [x_1, x_2]$.

A subscription is *stateful* if we cannot process it by simply ex-

*Contact author: badrish@cs.duke.edu

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
ACM 978-1-60558-102-6/08/06.

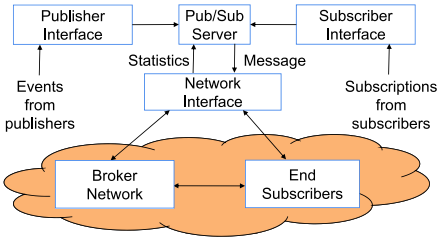

Figure 1: High-level design.

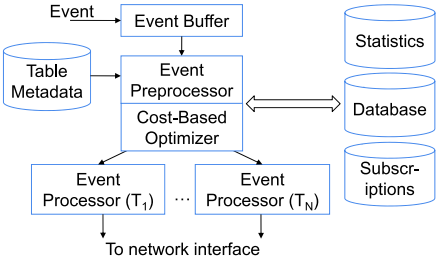

Figure 2: Server architecture.

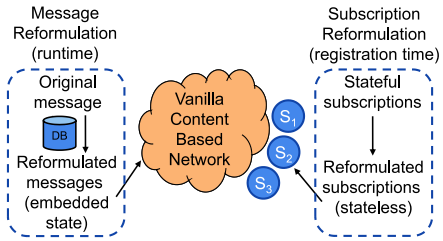

Figure 3: Reformulation.

amining the incoming event. Consider "`SELECT MIN(PER) FROM Stock WHERE` $x_1 \leq$ `Risk AND Risk` $\leq x_2$." This subscription is stateful because it may or may not be affected by an incoming event, depending on whether there is some other stock with `Risk` $\in [x_1, x_2]$ and `PER` lower than that of the event. Moreover, if an event raises the `PER` of a stock, subscriptions that previously had it as their minimum may need to look for new answers.

3 System Overview

Figure 1 shows the overall system design of ProSem. New subscriptions are registered with the *server* through a *subscriber interface*. Publishers deliver events to the server through a *publisher interface*. This interface also supports wrappers that poll Web pages or RSS feeds to generate events, as well as synthetic event generators for testing and experiments. The server processes events and uses the *network interface* to notify subscribers, either with direct IP unicast or through an overlay network of *brokers*. We rely only on standard, off-the-shelf network substrates that do not by themselves support application state; we use *reformulation* (Section 3.1) to support stateful subscriptions over a stateless network interface.

Figure 2 shows the architecture of the ProSem server. The server maintains the database (of events and related information) as well as the set of active subscriptions. ProSem groups subscriptions by *types*. For e.g., all range-min subscriptions with template "`SELECT MIN(PER) FROM Stock WHERE` $x_1 \leq$ `Risk AND Risk` $\leq x_2$" have the same type, where each instance is parameterized by (x_1, x_2). Dividing subscriptions into groups allows us to apply specialized indexing and group-prosemination techniques for each type.

ProSem reads each incoming event from an *event buffer* and sends the event to the *event preprocessor*, which determines (by analyzing schema and subscription templates) the set of subscription types potentially affected by the event. ProSem then uses a *cost-based optimizer* to choose the best prosemination strategy for each subscription type. The chosen strategies are then carried out by the *event processors* (specialized for each subscription type). After the event has been processed, the server updates the database and associated indexes accordingly, and moves on to the next event.

3.1 Prosemination Strategies

Stateless Subscriptions We support two alternatives for stateless subscriptions. First, the server can compute the affected subscriptions and unicast to each one. Second, we can use a *content-driven network (CN)* [5] deployed over a set of brokers, where each broker is responsible for notifying a subset of the subscriptions. CN directly supports stateless subscriptions: It allows message recipients to declare their interests to the network as predicates over message contents; senders simply inject messages into CN, which automatically forwards them to all recipients with matching interests.

Stateful Subscriptions For stateful subscriptions, we support several alternatives which vary in the way they interface server processing and network dissemination.

- *Enumeration*. The server computes the list of affected subscriptions along with notification contents for each of them. Notifications are then sent out by unicast. This approach is similar to the first alternative above for stateless subscriptions; nevertheless, handling a large number of stateful subscriptions at the server requires novel indexing and processing techniques.
- *Relaxation*. We relax stateful subscriptions into stateless ones to be handled by CN. Then, we rely on post processing at the brokers to compute updates to the original subscriptions. For e.g., a range-min subscription can be made stateless by dropping `MIN`. The downside is that maintaining the "larger" range subscriptions at each broker leads to many more notifications.
- *Reformulation*. Figure 3 illustrates reformulation. For each incoming event, the server dynamically reformulates it into messages embedded with state information, and then injects them into the network. Stateful subscriptions can then be reformulated into stateless ones over the reformulated messages, since the additional embedded state now allows these subscriptions to be processed using the message contents alone. The stateless reformulated subscriptions can be handled directly by a standard CN substrate.

Note that we have deliberately chosen a simpler system design for ProSem than other stateful systems that push processing of application state into the network, e.g., *SMILE* [7]. All prosemination strategies above feature clean server/network interfaces and rely on stateless off-the-shelf network components. We believe this design makes ProSem easier to deploy and maintain on a large scale.

While the strategies above are general, detailed techniques differ across different types of subscriptions. Reformulation in particular requires unconventional indexing and processing techniques, because it changes the role of processing from computing a list of affected subscriptions to reformulating messages, which are essentially *semantic descriptions* of affected subscriptions. ProSem uses reformulation techniques that were shown to outperform more conventional strategies for a variety of subscriptions [4, 5]. We show one example (range-min) below.

Example 1 (Reformulation for range-min subscriptions)**.** *Consider again the stateful subscriptions introduced in Section 2, which ask for minimum `PER` within `Risk` ranges of interest. Our reformulation is based on the concept of* maximum affected range (Mar)*, which intuitively captures an event's "extent of influence" on subscriptions. The* Mar *of a stock update event is the maximum `Risk` range in which no other stocks have a lower `PER`. Suppose an event decreases the `PER` of a stock with `Risk` $= x$ to y. We reformulate this event into the following message:*

$$\langle \texttt{NEW\_MIN} : y, \texttt{IN\_L} : x, \texttt{IN\_R} : x, \texttt{OUT\_L} : x_1, \texttt{OUT\_R} : x_2 \rangle,$$

where (x_1, x_2) is the Mar *of the updated stock. (For events that increase `PER`, see [4] for details.) On the other hand, we reformulate each range-min subscription over `Risk` range $[x_{\min}, x_{\max}]$ as a stateless filter over the reformulated message schema:*

$$(\texttt{OUT\_L} < x_{\min} \leq \texttt{IN\_L}) \wedge (\texttt{IN\_R} \leq x_{\max} < \texttt{OUT\_R}).$$

Upon receiving a message matching the reformulated predicate, a subscriber simply updates its minimum to `NEW_MIN`.

3.2 Run-Time Prosemination Optimization

ProSem draws parallels from query optimization in a traditional DBMS: alternative prosemination strategies are similar to alternative query execution plans in a DBMS. The available choices differ across subscription types, and depend on the available server indexes and network components. For each subscription type, the best choice may vary based on the incoming event being processed, runtime statistics, and system performance goals.

To provide flexibility and adaptivity, ProSem uses a cost-based optimizer to select a good *prosemination plan* for each incoming event at run-time. A plan specifies a prosemination strategy for each subscription type, and the order in which to execute them. We briefly describe how to cost each plans and find the best plan.

ProSem models the system cost under three basic categories of metrics. (1) *Throughput* is measured by the number of events per second that ProSem can support during continuous operation. It accounts for both *server processing time (SPT)* for computing outgoing notification messages and *server dissemination time (SDT)* for putting these messages on the network interfaces. (2) *Response time* is the delay from an event arrival until an affected subscription receives its notification. The average response time accounts for SPT, SDT, average propagation delay through the network, as well as the processing order of subscription types (those processed later experience longer response times). (3) *Bandwidth consumption* measures the total traffic incurred by ProSem between pairs of communicating nodes for notification dissemination. We define the *system cost* as a weighted combination of the three metrics above. Our goal is to choose a prosemination plan that minimizes system cost. ProSem allows system administrators to make dynamic adjustments to the objective weights.

To estimate the cost of a prosemination plan, ProSem continuously monitors statistics relevant to the three basic metrics. Another critical piece of information needed to optimize for each incoming event is the number of affected subscriptions for each subscription type. Recall that reformulation in effect computes semantic descriptions of affected subscriptions. Using these descriptions, ProSem can estimate the number of affected subscriptions without enumerating them—just like a DBMS estimates query result size—by maintaining summary statistics about subscriptions.

ProSem demonstrates a simple yet effective optimizer based on a greedy strategy that optimizes each subscription type separately. When deciding the processing order among subscription types, we give higher priority to those with lower per-subscription server costs, in order to reduce the overall average response time.

4 Demonstration Setup

We demonstrate ProSem using a local setup that provides a more controlled environment to showcase features. One or two laptops host the server and simulator-based versions of the network interface. Subscriptions register their interests with ProSem. On an incoming event, the server chooses the prosemination plan, processes the event, computes the outgoing messages, and disseminates them through selected network substrates. The network substrates (e.g., CN, unicast) perform dissemination using a simulator based on an INET-generated network topology.

Publisher Interface Publishers can directly send event updates to ProSem. We also use wrappers to gather data from websites, transform it, and send it to the server. The demo uses real and synthetic event traces to illustrate interesting trends.

Subscriber Interface Users use our customized client software to define subscriptions and receive notifications. In addition to "real" clients that demo observers can directly interact with, we also use large, synthetic subscriptions with simulated subscribers to demonstrate scalability.

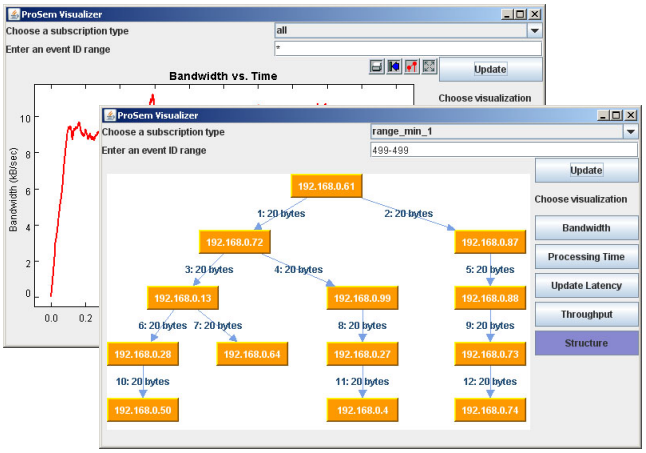

Figure 4: Visualizers.

Visualization We use our customized visualizers for the demo to show how dissemination occurs and how various choices affect prosemination costs. A *structure visualizer* shows the network topology and routes chosen for dissemination. It also shows the traffic along overlay links and summary statistics (such as number of subscriptions being handled) for each broker. The structure can be visualized for a particular event or a range of events, for a chosen subscription type. Figure 4 (front) is a screenshot based on data from our experiments with range-min subscriptions over CN. A *time-series visualizer* plots various performance metrics over time, including throughput, response time, and bandwidth consumption. Figure 4 (back) shows bandwidth usage from our experiments.

Demo Walkthrough We use a financial (stocks) application to demonstrate ProSem, with real and synthetic event traces. (1) We demonstrate support for stateless subscriptions as well as stateful ones such as range aggregation, select-joins, and select with value-based notification conditions. We show the process of registering subscriptions and receiving notifications. (2) To show how prosemination plans work and the advantages of novel reformulation-based prosemination strategies over simpler schemes, we use the structure visualizer to illustrate the details involved in processing and dissemination for several exemplifying events and subscription types. (3) To demonstrate scalability, we use the visualizers to compare the costs of prosemination plans for large subscription workloads. (4) To showcase the optimizer, we dynamically alter the optimization objective, to see how the optimizer adapts prosemination to different needs. We also show how the optimizer tailors prosemination at run-time for each event. The adaptation is clearly visible on our visualizers.

References

[1] G. Banavar et al. An efficient multicast protocol for content-based publish- subscribe systems. In *ICDCS*, 1999.

[2] A. Carzaniga et al. Design and evaluation of a wide-area event notification service. *ACM Trans. on Computer Systems*, 2001.

[3] M. Castro et al. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 2002.

[4] B. Chandramouli et al. On the database/network interface in large-scale publish/subscribe systems. In *SIGMOD*, 2006.

[5] B. Chandramouli et al. Value-based notification conditions in large-scale publish/subscribe systems. In *VLDB*, 2007.

[6] A. Demers et al. Towards expressive publish/subscribe systems. In *EDBT*, 2006.

[7] Y. Jin and R. Strom. Relational subscription middleware for internet-scale publish-subscribe. In *DEBS*, 2003.