PUBLISH/SUBSCRIBE OVER STREAMS

Yanlei Diao
Department of Computer Science
University of Massachusetts Amherst
yanlei@cs.umass.edu

Michael J. Franklin
Department of Electrical Engineering and Computer Science
University of California Berkeley
franklin@cs.berkeley.edu

SYNONYMS

Message-Oriented Middleware

DEFINITION

Publish/subscribe (pub/sub) is a many-to-many communication model that directs the flow of messages from senders to receivers based on receivers' data interests. In this model, publishers (i.e., senders) generate messages without knowing their receivers; subscribers (who are potential receivers) express their data interests, and are subsequently notified of the messages from a variety of publishers that match their interests.

HISTORICAL BACKGROUND

Distributed information systems usually adopt a three-layer architecture: a presentation layer at the top, a resource management layer at the bottom, and a *middleware layer* in between that integrates disparate information systems. Traditional middleware infrastructures are tightly coupled. Publish/Subscribe [Oki et al., 1993] was proposed to overcome many problems of tight coupling:

- With respect to communication, tightly coupled systems use static point-to-point connections (e.g., remote procedure call) between senders and receivers. In particular, a sender needs to know all its receivers before sending a piece of data. Such communication does not scale to large, dynamic systems where senders and receivers join and leave frequently. Pub/sub offers loose coupling of senders and receivers by allowing them to exchange data without knowing the operational status or even the existence of each other.
- With respect to content, tight coupling can occur in remote database access. To access a database, an application needs to have precise knowledge of the database *schema* (i.e., its structure and internal data types) and is at risk of breaking when the remote database schema changes. Extensible Markup Language (XML)-based pub/sub has emerged as a solution for loose coupling at the content level. Since XML is flexible, extensible, and self-describing, it is suitable for encoding data in a generic format that senders and receivers agree upon, hence allowing them to exchange data without knowing the data representation in individual systems.

In many pub/sub systems, message brokers serve as central exchange points for data sent between systems. Figure 1 illustrates a basic context in which a broker operates. Publishers provide information by creating streams of messages[1] that each contain a header describing application-specific information and a payload capturing the content of the message. Subscribers register their data interests with a message broker in a subscription language that the broker supports. Inside the broker, arriving subscriptions are stored as *continuous queries* that will be applied to all incoming messages. These

[1] Besides "messages", the words "events", "tuples", and "documents" are often used with similar meanings in various contexts in the database literature.

queries remain effective until they are explicitly deleted. Incoming messages are processed on-the-fly against all stored queries. For each message, the broker determines the set of queries matched by the message. A query result is created for each matched query and delivered to its subscriber in a timely fashion.

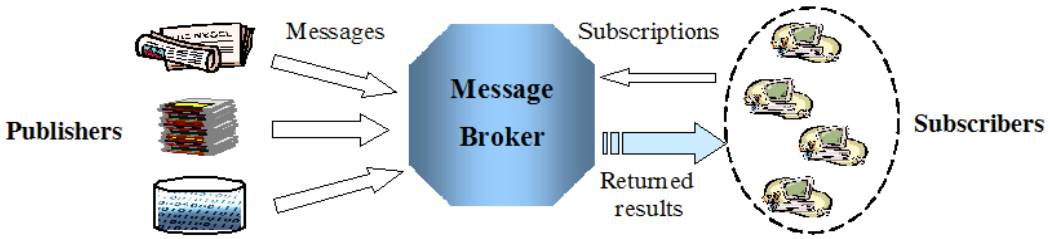

Figure 1: Overview of publish/subscribe

Figure 2 shows a design space for publish/subscribe over data streams. In this diagram, pub/sub systems are first classified by the data model and the query language that these systems support. Roughly speaking, there are three main categories.

- *Subject-based*: Publishers label each message with a subject from a pre-defined set (e.g., "stock quote") or hierarchy (e.g., "sports/golf"). Users subscribe to the messages in a particular subject. These queries can also contain a filter on the data fields of the message header to refine the set of relevant messages within a particular subject.
- *Complex predicate-based*: Some pub/sub systems model the message content (payload) as a set of attribute-value pairs, and allow user queries to contain predicates connected using "and" and "or" operators to specify constraints over values of the attributes. For example, a predicate-based query applied to the stock quotes can be "Symbol='ABC' and (Change > 1 or Volume > 50000)".
- *XML filtering and transformation*: Recent pub/sub systems have started to exploit the richness of XML-encoded messages, in particular, the hierarchical, flexible XML structure. User queries can be written using an existing XML query language such as XQuery. The rich XML structure and use of an XML query language enable potentially more accurate filtering of messages and further re-structuring of messages for customized result delivery.

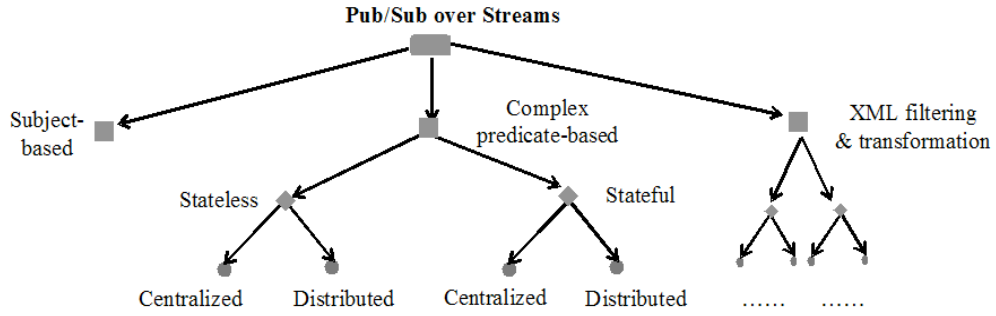

Figure 2: Design space of publish/subscriber over streams

Pub/sub systems can be further classified based on the style of query processing. In some systems, queries are applied only to individual messages, e.g., filtering messages, which does not involve any interaction across message boundaries. Such processing is referred to as *stateless*. Stateless processing is in contrast to stream query processing that maintains *state* over a long stream of messages, hence referred to as *stateful* processing. This distinction is illustrated for complex predicate-based systems in Figure 2.

Finally, pub/sub systems can be distinguished based on the distribution of the architecture, as also shown in Figure 2. In a coarse-grained fashion, this design space considers centralized and distributed processing. Distributed processing spreads the processing load for larger-scale pub/sub services; accordingly, it requires a more sophisticated routing functionality.

SCIENTIFIC FUNDAMENTALS

As with stream processing, subscriptions, stored as continuous queries inside a broker, need to be evaluated as data continuously arrives from other sources; that is, queries are evaluated every time when a new data item is received. Besides stream processing, pub/sub raises several additional challenges:

- *Scalability*. A key distinguishing requirement of pub/sub is scalability, in particular, in query population that pub/sub systems need to support. Such query populations can range from hundreds to millions in applications such as personalized content delivery. Given such populations, a salient issue is to efficiently search the huge set of queries to find those that can be matched by a message and to construct complete query results for them.
- *Robustness*. A second requirement of message brokers is the ability to perform in highly-dynamic environments where subscribers join and leave and their data interests change over time. Since message brokers see a constantly changing collection of queries, they must react quickly to query changes without adversely affecting the processing of incoming messages.
- *Distribution*. Due to the scale of message volume and query population, large-scale pub/sub may require the use of a network of message brokers to distribute the query population and message processing load. In this case, an additional issue is how to efficiently route a message from its publishing site to the set of brokers hosting relevant queries for complete query processing.

Scope of this article. The rest of the article focuses on complex predicate-based pub/sub systems. Pub/sub systems exploring XML filtering and transformation are described in detail in the entry "XML Publish/Subscribe".

Centralized, Stateless Publish/Subscribe

Le Subscribe [Fabret et al., 2001] and *Xlyeme* [Nguyen et al., 2001] are predicate-based message filtering systems that use centralized processing. In these systems, a predicate is a comparison between an attribute and a constant using relational operators such as '=', '>', and '<'. The main issue they address is how to efficiently match an incoming event, in the form of attribute value pairs, with the predicates of a large number of queries. The key idea is to *index* predicates as well as to *cluster* queries. In particular, *Le Subscribe* uses multi-attribute hash indexes to evaluate several predicates in a query with a single operation. In addition, it groups queries based on the number of contained predicates and the common conjunction of equality predicates, so many queries can be (partly) evaluated using a single operation. It further offers cost-based algorithms to find optimal clustering and to dynamically adjust it.

Centralized, Stateful Publish/Subscribe

NiagaraCQ [Chen et al., 2000] considers continuous queries with more complex predicates that can compare attributes of an input message to constants or to attributes of another message. To efficiently handle multiple queries, it groups query plans of continuous queries based on common *expression signatures*: an expression signature presents the same syntax structure, but possibly different constant values, in different queries. Consider queries that are interested in stock quotes of different symbols. Traditional query processing involves repeated retrieval of the symbol attribute from input and evaluation of different predicates on this attribute for different queries. The group plan employs a constant table to store the constant values form different queries, and retrieves this attribute from the input once and then performs an equality join of the retrieved value and the constant table to find all matching queries. For robust processing, NiagaraCQ constructs group plans incrementally: Given a new query, it constructs the query plan, and merges the query plan bottom up with a group plan with the same signature, extending the

group plan with the mismatched branch(es) if necessary. This process is incremental as the addition of the new query plan does not affect existing queries.

Recently, there has been a significant amount of activity on handling continuous and time-varying tuple streams, resulting in the development of multiple general-purpose stream management systems [Abadi et al., 2003; Chandrasekaran et al, 2003; Motwani et al., 2003]. These systems support complex continuous queries that join multiple streams and/or compute aggregate values over a period of time called a *window* (hence, performing stateful processing). While this surge of research explores a broad set of issues such as adaptivity and approximation, shared processing of window-based queries [Chandrasekaran et al, 2003; Motwani et al., 2003; Krishnamurthy 2006] is of particular relevance to pub/sub. (The reader is referred to the articles on stream processing for further details of these techniques.)

Several special-purpose pub/sub systems have been recently proposed to handle temporal correlations among events in a stream. SASE [Wu et al., 2006] supports sequencing operators that integrate parameterized predicates (i.e., predicates that compare different events), negation, and windowing. It explores a new query processing abstraction that uses an automaton-based implementation for fast sequence operations and relational-style post-processing for other tasks such as negation and windowing. It also devises a set of optimizations in this automaton-based framework for efficiency and scalability. Cayuga [Demers et al. 2006] offers an algebra for expressing event sequences that may address a finite yet unbounded number of events with a similar property, and employs a more sophisticated automaton model to support this algebra. Its implementation focuses on multi-query optimization including merging states of automata for different queries and further indexing query predicates.

Distributed, Stateless Publish/Subscribe

In distributed pub/sub systems, messages are published and subscriptions are registered to different brokers. A key issue is to efficiently route each message from its publishing site to the subset of brokers hosting relevant queries for complete query processing. For complexity reasons, most distributed pub/sub systems restrict themselves to stateless services.

ONYX [Diao et al., 2004] presents an overview of a pub/sub network exploring *content-based routing*. In this paradigm, brokers are organized as an application-level overlay network with a particular topology. When a new message enters the broker network, the root broker as well as each intermediate broker routes the message to its neighbouring brokers based on the correspondence between the content of the message and the subscriptions residing at and reachable from those brokers. Content-based routing is used as a key mechanism to avoid the flooding of messages to all brokers in the network, hence reducing bandwidth usage and broker processing load and rendering better scalability.

ONYX consists of two layers of functionality. The lower layer deals with the overlay network; in particular, for each broker, it constructs a broadcast tree that is rooted at that broker and reaches all other brokers in the network. On top of these broadcast trees, the higher layer performs content-based processing by dealing with subscriptions and messages. Two issues determine the effectiveness of content-based routing. The first is how to partition subscriptions and assign them to host brokers. Results of ONYX show that content-based routing is most effective if the clustering of subscriptions results in mutual exclusiveness in data interest among host brokers. The second issue is how to aggregate subscriptions from their host brokers and place such aggregations as routing specifications in the intermediate brokers for later directing the message flow. Different degrees of generalization are possible depending on the precision-efficiency tradeoff suitable for each pub/sub system.

For content-based routing, Gryphon [Aguilera et al., 1999] and Siena [Carzaniga and Wolf, 2003] both aggregate subscriptions into compact, precise in-network data structures and use efficient algorithms to search these data structures to determine the routing of the messages to other brokers in the network.

XPORT [Papaemmanouil et al., 2006] focuses on the construction, maintenance, and optimization of an overlay dissemination tree of the available message brokers in the system. Its tree-oriented optimization framework consists of a generic aggregation model that allows system cost to be expressed through various combinations of aggregation functions and local metrics, and distributed iterative optimization protocols for cost-based optimization of the overlay structure.

Distributed, Stateful Publish/Subscribe

For stateful publish/subscribe, Chandramouli et al. [Chandramouli et al., 2006] adopt the following model: Users define subscriptions as SQL views over a conceptual (possibly distributed) database and messages are published as updates to the database; if a database update affects a subscription, the pub/sub system sends a notification to the subscriber containing the change to the content of the subscription view. The main idea of this work is to explore appropriate interfacing between the database and the pub/sub network so that existing stateless pub/sub networks can be leveraged for efficient dissemination. To do so, the key is to transform published messages into a semantic description of affected subscriptions (performed at the database side) and subscriptions into a predicate over the semantic description (evaluated in the stateless pub/sub network). Consider a selection-join subscription $\sigma_p(\sigma_{p\text{-}R}\ R \rhd\lhd \sigma_{p\text{-}S}\ S)$. If a new message applies an update ΔR to table R, its effect on the subscription, $\sigma_p(\sigma_{p\text{-}R}\ \Delta R \rhd\lhd \sigma_{p\text{-}S}\ S)$, requires access to table S that is not in the original update message (hence, stateful processing). The proposed solution reformulates each message ΔR into a series of messages containing the tuples in $\Delta R \rhd\lhd S$ at the database side; to utilize a stateless pub/sub network, it transforms the select-join subscription into a simple condition that evaluates $\sigma_{p \wedge p\text{-}R \wedge p\text{-}S}$ over reformulated messages.

KEY APPLICATIONS

Personalized content delivery. This class of applications provide personalized filtering and delivery of news feeds, web feeds (RSS), stock tickers, sport tickers, etc. to large numbers of online users.

Online auction and online procurement. Through these applications, users can create their own feeds for their favorite searches, for example, on eBay.

Enterprise information management. Pub/sub has been traditionally used to support application integration, which integrates disparate, independently-developed applications into new services via the loose coupling of senders and receivers based on receivers' data interest.

System and network monitoring. Pub/sub has been recently used in system and network monitoring, where large-scale complex systems generate reports categorizing various aspects of system performance and resource utilization, and system administrators, end users, and visualization applications subscribe to receive updates on particular aspects of those reports.

DATA SETS

Many data sources are available online, including RSS feeds (indicated by the orange button labeled with "RSS" or "XML" in many web pages) and financial feeds (e.g., Yahoo! Finance).

CROSS REFERENCES

Stream processing, Continuous queries, XML, XML document, XPath/XQuery, XML publish/subscribe

RECOMMENDED READING

[Abadi et al., 2003] Abadi, D., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. Aurora: A New Model and Architecture for Data Stream Management. In *VLDB Journal*, 12(2), 120-139, August 2003.

[Aguilera et al., 1999] Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., and Chandra, T.D. Matching Events in a Content-Based Subscription System. In *Proc. of Principles of Distributed Computing* (*PODC'99*), Atlanta, GA, May 1999.

[Carzaniga and Wolf, 2003] Carzaniga, A., and Wolf, A.L. Forwarding in a Content-Based Network. In *SIGCOMM*, 163-174, Karlsruhe, Germany, August 2003.

[Chandramouli et al., 2006] Chandramouli, B., Xie, J., and Yang, J. On the database/network interface in large-scale publish/subscribe systems. In *SIGMOD*, 587-598, 2006.

[Chandrasekaran et al, 2003] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., and Shah, M.A. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, Asilomar, CA, January 2003.

[Chen et al., 2000] Chen, J., Dewitt, D.J., Tian, F., and Wang, Y. NiagaraCQ: A scalable continuous query system for Internet databases. In *SIGMOD*, 379-390, Dallas, Texas, May, 2000.

[Demers et al., 2006] Demers, A.J., Gehrke, J., Hong, M., Riedewald, M., and White, W.M. Towards Expressive Publish/Subscribe Systems. In *EDBT*, 627-644, 2006.

[Diao et al., 2004] Diao, Y., Rizvi, S., and Franklin, M.J. Towards an Internet-Scale XML Dissemination Service. In *VLDB*, 612-623, August 2004.

[Fabret et al., 2001] Fabret, F., Jacobsen, H.A., Llirbat, F., Pereira, J., Ross, K.A., and Shasha, D. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. In *SIGMOD*, 115-126, May 2001.

[Krishnamurthy 2006] Krishnamurthy, S. Shared Query Processing in Data Streaming Systems. Ph.D. Dissertation, University of California, Berkeley.

[Motwani et al., 2003] Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., and Varma, R. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *CIDR*, Asilomar, CA, January 2003.

[Nguyen et al., 2001] Nguyen, B., Abiteboul, S., Cobena, G., and Preda, M. Monitoring XML data on the Web. In *SIGMOD*, 437-448, Santa Barbara, May 2001.

[Oki et al., 1993] Oki, B., Pfleugl, M., Siegel, A., and Skeen, D. The Information Bus: An Architecture for Extensible Distributed System. In *SOSP*, 58-68, December 1993.

[Papaemmanouil et al., 2006] Papaemmanouil, O., Ahmad, Y., Çetintemel, U., Jannotti, J., and Yildirim, Y. Extensible optimization in overlay dissemination trees. In *SIGMOD*, 611-622, 2006.

[Wu et al., 2006] Wu, E., Diao, Y., and Rizvi S. High-performance complex event processing over streams. In *SIGMOD*, 407-418, 2006.