Historic Data Access in Publish/Subscribe

G. Li A. Cheung Sh. Hou S. Hu V. Muthusamy
R. Sherafat A. Wun H.-A. Jacobsen S. Manovski
Middleware System Research Group
University of Toronto, Canada

ABSTRACT

We develop a content-based publish/subscribe platform, called PADRES, which is a distributed middleware platform with features inspired by the requirements of workflow management and business process execution. These features constitute original additions to publish/subscribe systems and include an expressive subscription language, historic, query-based data access, composite subscription processing, a rule-based matching and routing mechanism, and the support for the decentralized execution of service-oriented applications.

Categories and Subject Descriptors

H.4.1 [**Information Systems**]: Information Systems Applications

General Terms

Design, Management, Experimentation

Keywords

publish/subscribe, content-based routing, query routing, composite subscriptions, database

1. INTRODUCTION

The PADRES project [1] aims to create a distributed content-based publish/subscribe (p/s) system. The project is a collaboration between the Middleware System Research Group at the University of Toronto, CA and Sun Microsystems.

The primary research focus of PADRES is applying and extending the content-based p/s paradigm to fit the requirements of workflow management and business process execution in a large-scale distributed environment (e.g., hundreds of nodes, thousands of users) [8]. Specifically, PADRES is seen as a message-oriented middleware layer for such a system, although some non-typical middleware features such as historic data access are included. A secondary goal of PADRES is to seamlessly connect centralized messaging systems. By providing bindings for various messaging formats, other messaging systems can be connected as PADRES clients and can communicate with each other. PADRES, in effect, can *federate* disparate messaging systems.

The workflow management context introduces several challenges in the use of content-based p/s. Several requirements are not satisfied by traditional content-based p/s systems. Seamless access to historic data, a powerful subscription language and matching engine, highly scalable message routing, and flexible network topologies are several of the key features of PADRES needed to meet these requirements.

The PADRES subscription language is based on the traditional `[attribute, operator, value]` predicates used in several existing content-based p/s systems [7, 4, 15, 14]. Each message has a mandatory tuple describing the `class` of the message and an arbitrary number of additional tuples specifying the message content. The `class` attribute provides a guaranteed selective predicate similar to the topic in topic-based p/s systems. The `class` concept could potentially be extended to a full semantic ontology [17, 18]. This traditional language is augmented by the use of composite subscriptions [13]. Composite subscriptions combine several atomic subscriptions, and are only matched after all the contained subscriptions are individually matched. The matching engine in PADRES is built using a Rete-based rule engine [9]. The rule engine is a powerful method of performing complex content-based matching, and naturally supports composite subscriptions.

Historic data access for content-based p/s systems is the focus of this demonstration. PADRES, unlike existing content-based p/s systems, allows the user to subscribe to data published in both the future and the past. Accessing both future and past publications using standard subscription semantics is a novel and useful approach in the distributed p/s context. Databases, associated with brokers in the network, store publications as they are published. Later, upon receiving a request for the historic data, the brokers re-publish relevant publications from their databases. Publications are replicated across several databases in order to improve data availability. Databases can sink a part of or the entire publication space.

Note that in the content-based p/s paradigm, no direct address information of participants is available, so all publi-

DEBS '07, June 20-22, 2007 Toronto, Ontario, Canada

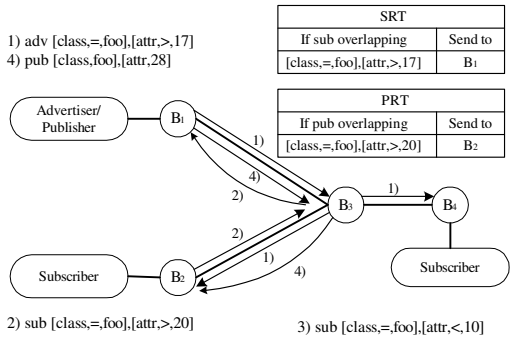

Figure 1: PADRES Network Architecture

cations and subscriptions must be routed according to their content. This fact precludes directly querying the distributed databases for past data. In fact, it is *impossible* for the client to know which databases to query. The content-based routing is the fundamental difference between p/s networks and traditional IP-routed networks.

In order to provide historic data access, PADRES supports time-based subscriptions. Time-based subscriptions contain a time interval (open or closed) that indicates the desired set of publications. Queries are broken down into future (traditional p/s) and historic portions, and the historic subscriptions are routed to the appropriate databases.

2. SYSTEM DESCRIPTION

The PADRES system consists of a set of p/s brokers connected by a peer-to-peer overlay network. Clients connect to brokers using various binding interfaces such as Java Remote Method Invocation (RMI) and Java Messaging Service (JMS). PADRES is implemented in Java, and uses RMI as its native transport protocol. There is an administrative client which allows a user to visualize the network and control the brokers in the system.

2.1 Network Architecture

The overlay network connecting the brokers is a set of RMI connections that form the basis for message routing. The overlay routing data is stored in Overlay Routing Tables (ORT) on each broker. Message routing in PADRES is based on the publish-subscribe-advertise model established by the SIENA project [4]. We assume that publications are the most common messages, and that advertisements are the least common.

Advertisements are effectively flooded to all brokers along the overlay network using the ORT. The set of advertisements seen by a broker are used to construct the Subscription Routing Table (SRT) for that broker. The SRT is essentially a list of `[advertisement content specification, last hop]` tuples, although it is actually implemented using a Rete in our rule engine. Clients are required to advertise before publishing data, but may subscribe at any time.

In order to reduce network traffic and decrease routing table sizes, brokers merge advertisements before forwarding them. Advertisements with significant overlap are propagated as a single, more general advertisement. The PADRES merging scheme is an extension of the covering concept in SIENA [4], and is similar to subscription merging in Rebeca [15].

Subscriptions are routed hop by hop according to the SRT at each broker. The set of subscriptions seen by a broker are used to construct the Publication Routing Table (PRT) for that broker. Like the SRT, the PRT is logically a list of `[subscription content specification, last hop]` tuples implemented using a Rete. Subscriptions are merged in the same manner as advertisements.

Fig. 1 shows the overlay network, SRT and PRT. In this figure, an advertisement *1)* is propagated from B_1. Then a matching subscription *2)* enters from B_2, and is routed along the SRT. For instance, *2)* overlaps *1)* at broker B_3, according to the SRT of B_3, it is sent to B_1. Subscription *3)* does not overlap advertisement *1)*, so it is not forwarded by B_4. Lastly, publication *4)* matching subscription *2)* is routed along the PRT formed by *2)* to B_2.

The PADRES broker overlay can be either acyclic or cyclic. A cyclic overlay improves the robustness and failure resilience of the messaging substrate by providing alternate routing paths between subscribers and publishers. In the cyclic case, publications can be routed to subscribers using the "best" path, dynamically adapting to network traffic and failures.

2.2 Broker Architecture

The PADRES brokers are modular software components built on a set of queues. Each queue represents a unique message destination, and has a message handler assigned to it. The message handlers are the matching engine, broker controller and client bindings. A diagram of the broker internals is provided in Fig. 2.

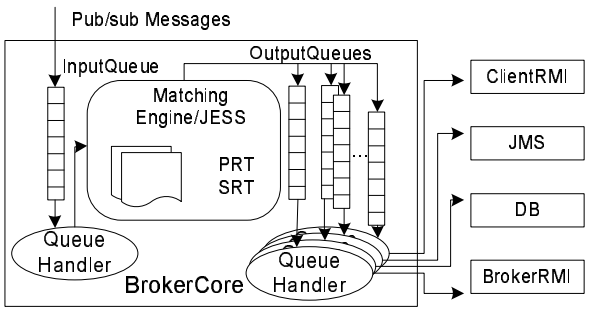

Figure 2: PADRES Broker Internals

2.3 Historic Data Access Description

The PADRES historic data access scheme was chosen primarily for its transparency and flexibility. Unlike a standard subscription, a historic subscription has a specified time range that is not in the future, and contains a special attribute which guarantees that historic data will not be received by regular subscribers whose subscriptions overlap the historic one. All operators allowed in standard subscriptions are also supported in historic subscriptions.

Historic databases are attached to brokers as clients using a database binding as shown in Fig 2. Each database issues a subscription to its broker with appropriate {`[class, =,`

DB_CONTROL], [DatabaseID, =, DB1]} predicates. A user can cause a database to begin storing certain publications by issuing a {[class, DB_CONTROL],[DatabaseID, DB1], [cmd, STORE], [content, '[class, =, foo], [attr, >, 20]']} command (as a publication). The database then advertises {[class, =, foo], [attr, >, 20]} in anticipation of publishing historic events, and begins storing matching publications, such as {[class, foo], [attr, 28]} (converting them to appropriate SQL INSERT statements first). If the user later (after 12:00) subscribes to {[class, =, foo], [time, >, 0800], [time, <, 1200]}, the database will receive the subscription (since it matches the database's advertisement), perform an SQL SELECT for the appropriate publications, and re-publish the stored publications in the time interval (08:00, 12:00).

With historic data access functionality, PADRES allows clients to subscribe to future data, historic data, and hybrid data (a combination of future and historic data). Both atomic subscriptions and composite subscriptions (which are used to express correlations) can be used to access future, historic, or hybrid data.

Since all messages are routed using a content-based algorithm, it is impossible to directly query the databases for past data. In fact, it is not even possible for clients to know which databases to query. Instead, since the historic data access functionality is built entirely over p/s semantics, the PADRES routing algorithm will automatically route queries to the appropriate databases along the most efficient path.

There are many research challenges inherent in this historic data access scheme. The placement of historic databases, assignment of content specification to each database, and avoiding duplicate historic publications are ongoing research challenges.

3. RELATED WORK

There are several existing p/s systems that share some of the features of PADRES. The PADRES content-based routing protocol is similar to the protocol used in SIENA [4], extended with the subscription merging concept of Rebeca [15]. PADRES relies on mandatory advertisements to optimize subscription propagation. The PADRES matching engine is built using the Rete-based rule engine from the Java Expert System Shell [10]. There is virtually nothing in the literature about using the Rete algorithm or a rule engine in this manner. The historic data access methods in PADRES is unique in the p/s context. There is some work on propagating historic publications, but only in the limited context of mobile clients [6, 16], where historic data is retained by either requiring each publisher to store its own publications, or by having brokers manage the historic data based on timestamps. PADRES proposes a more flexible storage strategy, supporting the ability to easily assign arbitrary portions of the data space (specified using content-based subscriptions) to each database.

Composite subscriptions are the p/s equivalent of composite events. Composite events are an important concept for the network management application context and have not received much attention in the p/s literature [19]. An instance of a composite event is formed by a temporal or causal combination of constituent event instances. For example, the composite event $E_C = (e_1; e_2)_t$ uses a sequence operator to specify that if event e_1 is satisfied before e_2 within time duration t, the composite event E_C is satisfied. In the context of p/s this could be cast into a subscription – a *composite subscription* [13] — concerned with the temporal state of the matched subscription expressions. For example, a subscription S_C can be defined as $(s_1; s_2)_t$, where s_1 and s_2 are conventional expressions comprising predicates. The semantic of such a subscription is that the subscriber wants to be notified only if events e_1 and e_2 occur within the time duration t such that e_1 satisfies s_1 and e_2 satisfies s_2. Composite events are described in [11, 5]. CEA [19] is a composite event detection framework built as an extension of an existing publish/subscribe middleware platform, whereas PADRES builds the composite subscription processing and composite event detection capability directly into the publish/subscribe layer.

Similar problems are addressed in stream-based distributed applications [12]. However, one difference is that records in data streams follow the same schema, while publications may not have this property. Moreover, publications may come from multiple data sources while a data stream has one source. Exploring how data streams are processed in a distributed environment may give us some insight for publication processing, especially for historic and composite subscriptions.

4. SOFTWARE DEMONSTRATION

The objective of the software demonstration is to show the PADRES system in operation. We will demonstrate a retail application management scenario. This scenario demonstrates a cyclic content-based p/s network with two databases federated in the system. Customers and sales managers are p/s clients who can publish and subscribe. This scenario will highlight the benefits of composite, historic, and hybrid subscriptions in the retail application context. The demonstration topology is shown in Fig. 3.

PADRES also provides a graphical interface which allows a user or developer to visualize the network topology and the message routing. The monitoring tool interacts with the broker network by publishing and subscribing to messages. It does not have a special role and can be connected to any broker in the network. An example of the graphical PADRES monitor in Fig. 4 shows a 100-broker overlay running on PlanetLab [2].

4.1 Retail Management Scenario

In this scenario customers publish information such as purchase orders, item returns, and cancellations. A customer order, for example, may include attributes like the order ID, customer ID, product name, price, user type[1], order time, etc. For example, a publication to place an order may look like {[class, Order], [OrderID, C100-01], [CustomerID, C100], [Item, shoes], [UserType, gold], [Time,'Apr 3 07:33:21 2007']}. Also, the machines in the retail system report their status (overloaded, healthy, etc.) with publications such as {[class,SysInfo], [Status,overload],

[1]The user type includes general users, silver users, gold users, and VIP users.

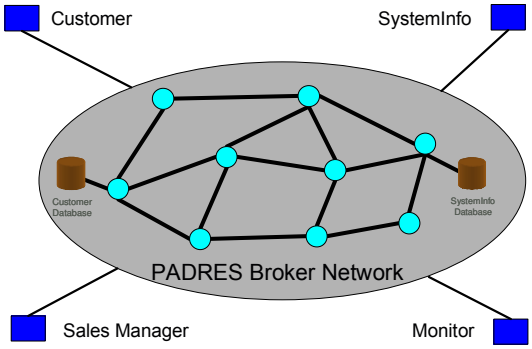

Figure 3: PADRES Demo Architecture

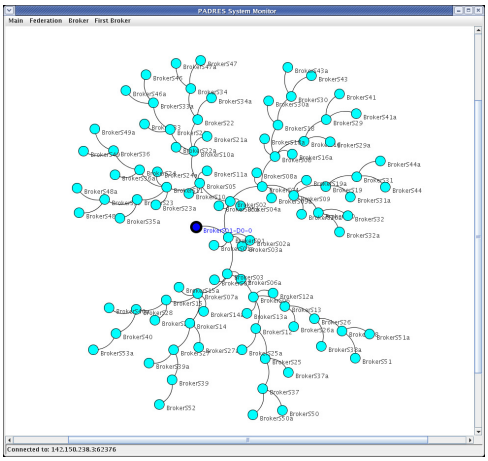

Figure 4: PADRES System Monitor

`[Time,'Apr 3 09:00:00 2007']}`. Therefore, publication streams are produced by customers and server status reporting agents. A publication propagates from the publishing client to its connecting broker and then into the network based on the PRT built from the subscriptions in the network. The routing of publications can be verified by notifications received by the clients who have issued subscriptions that match the published data.

Databases act as subscribers and can be located at any broker in the system. The databases sink the matching publications for later access by sales managers, for auditing purposes, archival, or any other use. The sales manager issues regular subscriptions to be notified of future customer orders and system information, historic subscriptions to retrieve past publications, hybrid subscriptions to access both past and future publications, and composite subscriptions to express correlations between data streams. The results of these queries are displayed live in the sales manager GUI. In another visualization, the start-up and materialization of the overlay network can be observed from the monitor GUI.

Subscribe to future orders: A sales manager can express interest in orders placed in the future for specific items using a subscription such as {`[class, eq, Order]`, `[OrderID, isPresent, any]`[2], `[CustomerID, isPresent, any]`, `[Item, eq, shoes]`, `[UserType, isPresent, any]`}. The subscription will propagate from the sales manager client to its connecting broker and then into the network based on the SRT formed using publishers' advertisements. Matching publications (e.g., orders for shoes) issued after the subscription will be routed to the sales manager. In addition to displaying the set of matching publications in the sales manager GUI, the publication propagation is verified using a tracing feature in the monitoring GUI that allows messages to be tracked live as they traverse the network.

Subscribe to historic orders: A sales manager that wants to retrieve orders placed in the past would issue a subscription such as {`[class, eq, Order]`, `[OrderID, isPresent, any]`, `[CustomerID, isPresent, any]`, `[Time, <, now`[3]`]`}. A subscription for historic data is propagated through the network to one or more databases that can execute the query and the relevant data is propagated back to the subscribing client. The query is routed through the network in the same manner as a subscription for future data. Notably, since no information about the databases' addresses are available to the client, the query is propagated along the SRT. The historic orders re-published from databases are only received by the subscriber, and other clients who may have issued overlapping queries will not receive duplicate orders. The effect of the query can also be observed from the sales manager GUI as it plays back the resulting data.

Subscribe to both future and historic orders: Orders placed by a particular customer (in the past and in the future) may also be interesting to the sales manager. Such hybrid subscriptions are split into two parts by brokers: a query on the historic data and a subscription for the future data, effectively mapping to the above two cases.

Subscribe to correlated data: If a server becomes overloaded, a sales manager may wish to manually review orders affected by the overload server. In this scenario, each server is assigned to handle orders from one or more user types, and so a subscription to express interest in orders handled by overloaded servers is achieved with a composite subscription such as {{`[class, eq, Order]`, `[OrderID, isPresent, any]`, `[CustomerID, isPresent, any]`, `[Time, isPresent, any]`}, `[UserType, eq, $X`[4]`]` & {`[class,eq, SysInfo]`, `[UserType, eq, $X]`}}. This composite subscription is satisfied by two independent publication streams, and is managed at an edge broker where it is split into its constituent atomic subscriptions, which are then propagated through the network in the usual manner. While the composite subscription above only matches future data, it is also possible to express such correlations against future, historic, or hybrid data streams. The matching state of a composite subscription is evaluated at the edge broker who notifies

[2]The special operator isPresent means an attribute could be any value in a given range.

[3]The token *now* evaluates to the time when the query is issued.

[4]Variables are bound to values in matching publications.

the client of correlated matches. As with the above subscriptions, composite subscription matching can be demonstrated using the client and monitor GUIs.

4.2 PADRES Matching Performance

We compare the rule-based matching engine implemented in JESS with two other methods. One is a naive matching algorithm which linearly scans the routing table to find the matched subscriptions. The other is a matching algorithm that is similar to the predicate counting algorithm [3]. This algorithm calculates distinct predicates only once. Our experiments show that the rule-based matching engine using a Rete network is very efficient. It takes only 4.52ms to match a publication against 200,000 subscriptions. This is 80 times and 20 times faster that the naive approach and the counting algorithm, respectively. The matching performance indicates that the PADRES engine is suitable for large scale p/s systems and can process a large number of publication and subscription messages efficiently.

5. ACKNOWLEDGEMENTS

This research was funded in part by CFI, OIT,OCE, NSERC, CA and Sun. We would like to thank former PADRES members Eli Filder, Pengcheng Wan and Matt Medland for their efforts.

6. REFERENCES

[1] `http://padres.msrg.toronto.edu`.
[2] `https://www.planet-lab.org/`.
[3] G. Ashayer, H. Leung, and H.-A. Jacobsen. Predicate matching and subscription matching in publish/subscribe systems. In *DEBS'02 Workshop at ICDCS'02*, Vienna, Austria, 2002.
[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.
[5] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data and Knowledge Engineering*, 14(1):1–26, 1994.
[6] M. Cilia, L. Fiege, C.Haul, A.Zeidler, and A. P. Buchmann. Looking into the Past: Enhancing Mobile Publish/Subscribe Middleware. In *Proceedings of 2nd International Workshop on Distributed Event-Based Systems (DEBS'03)*. IEEE Computer Society, 2003.
[7] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 30(2):115–126, 2001.
[8] E. Fidler, H.-A. Jacobsen, G. Li, , and S. Mankovski. Distributed publish/subscribe for workflow management. *International Conference on Feature Interactions in Telecommunications and Software Systems (ICFI'05), Leisester, UK*, 2005.
[9] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, 1982.
[10] E. J. Friedman-Hill. Jess, The Rule Engine for the Java Platform. `http://herzberg.ca.sandia.gov/jess/`.
[11] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite Event Specification in Active Databases: Model & Implementation. In *Proceedings of the 18th International Conference on Very Large Databases (VLBD92)*, 1992.
[12] V. Kumar, Z. Cai, B. F. Cooper, and G. Eisenhauer. IFLOW: Resource-aware overlays for composing and managing distributed information flows. In *EuroSys*, 2006.
[13] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *ACM/IFIP/USENIX 6th International Middleware Conference, Grenoble, France*, 2005.
[14] H. Liu and H.-A. Jacobsen. A-topss: A publish/subscribe system supporting imperfect information processing.
[15] G. Mühl. Generic constraints for content-based publish/subscribe systems. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Proceedings of the 6th International Conference on Cooperative Information Systems (CoopIS '01)*, volume 2172 of *LNCS*, pages 211–225, Trento, Italy, 2001. Springer-Verlag.
[16] G. Mühl, A. Ulbrich, K. Herrmann, and T. Weis. Disseminating information to mobile clients using publish-subscribe. *IEEE Internet Computing*, 8(3):46–53, 2004.
[17] M. Petrovic, I. Burcea, and H.-A. Jacobsen. S-ToPSS - A Semantic Publish/Subscribe System. In *Proceedings of the 29th International Conference on Very Large Databases*, Berlin, Germany, September 2003.
[18] M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS: fast filtering of graph-based metadata. In *WWW*, pages 539–547, New York, NY, USA, 2005. ACM Press.
[19] P. R. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *IEEE Network Magazine, Special Issue on Middleware Technologies for Future Communication Networks*, January/February 2004.