

# The PADRES Publish/Subscribe System<sup>1</sup>

**Hans-Arno Jacobsen, Alex Cheung, Guoli Li, Balasubramaneyam Maniymaran, Vinod Muthusamy, Reza Sherafat Kazemzadeh**

*Middleware Systems Research Group, University of Toronto, Canada*

## KEYWORDS

Publish/Subscribe, Content-based Routing, Composite Subscription, Historic Data Access, Load Balancing, Fault-tolerance

## ABSTRACT

This chapter introduces PADRES, the publish/subscribe model with the capability to correlate events, uniformly access data produced in the past and future, balance the traffic load among brokers, and handle network failures. The new model can filter, aggregate, correlate and project any combination of historic and future data. A flexible architecture is proposed consisting of distributed and replicated data repositories that can be provisioned in ways to tradeoff availability, storage overhead, query overhead, query delay, load distribution, parallelism, redundancy and locality. This chapter gives a detailed overview of the PADRES content-based publish/subscribe system. Several applications are presented in detail that can benefit from the content-based nature of the publish/subscribe paradigm and take advantage of its scalability and robustness features. A list of example applications are discussed that can benefit from the content-based nature of publish/subscribe paradigm and take advantage of its scalability and robustness features.

## 1.1 INTRODUCTION

The publish/subscribe paradigm provides a simple and effective method for disseminating data while maintaining a clean decoupling of data sources and sinks (Cugola, 2001; Fabret, 2001; Castro, 2002; Fiege, 2002; Carzaniga, 2003; Eugster, 2003; Li, 2005; Ostrowski, 2006; Rose, 2007). This decoupling can enable the design of large, distributed, and loosely coupled systems that interoperate through simple publish and subscribe invocations. While there are many applications such as information dissemination (Liu, 2004; Nayate, 2004; Liu, 2005) based on group communication (Birman, 1999) and topic-based publish/subscribe protocols (Castro, 2002; Ostrowski, 2006), a large variety of emerging applications benefit from the expressiveness, filtering, distributed event correlation, and complex event processing capabilities of *content-based publish/subscribe systems*. These applications include RSS feed filtering (Rose, 2007), stock-market monitoring engines (Tock, 2005), system and network management and monitoring (Mukherjee, 1994; Fawcett, 1999), algorithmic trading with complex event processing (Keonig, 2007), business process management and execution (Schuler, 2001; Andrews, 2003;), business activity monitoring (Fawcett, 1999), workflow management (Cugola, 2001), and service discovery (Hu, 2008).

Typically, a distributed content-based publish/subscribe systems is built as an application-level overlay of content-based publish/subscribe brokers, with publishing data sources and

---

<sup>1</sup> This paper will be published as a chapter in “Handbook of research on advanced distributed event-based systems, publish/subscribe and message filtering technologies”.

subscribing data sinks connecting to the broker overlay as clients. In a content-based publish/subscribe system, message routing decisions are not based on destination IP-addresses but on the content of messages and the locations of data sinks that have expressed an interest in that content.

To make the publish/subscribe paradigm a viable solution for the above applications, additional features must be added. This includes support for *composite subscriptions* to model and detect composite events, and to enable event correlation and in-network event filtering to reduce the amount of data transferred across the network.

Furthermore, the publish/subscribe substrate that carries and delivers messages must be robust against non-uniform workloads, node failures, and network congestions. In PADRES<sup>2</sup>, robustness is achieved by supporting alternate message routing paths, load balancing techniques to distribute load, and fault resilience techniques to react to broker failures.

It is also essential for a publish/subscribe system to provide tools to perform monitoring, deployment, and management tasks. Monitoring is required throughout the system to oversee the actual message routing, the operation of content-based brokers, and the interaction of applications via the publish/subscribe substrate. Deployment support is required to bring up large broker federations, orchestrate composite applications, support composition of services and business processes, and to conduct controlled experiments. Management support is required to inspect and control live brokers.

This chapter presents the PADRES content-based publish/subscribe system developed by the *Middleware Systems Research Group* at the University of Toronto. The PADRES system incorporates many unique features that address the above concerns and thereby enable a broad class of applications. The remainder of this chapter begins with a description of the PADRES language model, network architecture and routing protocol in Section 1.2. This is followed, in Section 1.3, by an outline of the PADRES load balancing capabilities whereby the system can automatically relocate subscribers in order to avoid processing or routing hotspots among the network of brokers. Section 1.4 then addresses failure resilience describing how the PADRES routing protocols are able to guarantee message delivery despite a configurable number of concurrent crash-stop node failures. Some of the PADRES distributed management features are presented in Section 1.5, including topology monitoring and deployment tools. Next, Section 1.6 discusses a wide variety of applications and illustrates how the features of the PADRES system enable or support the development of these applications. Finally, a survey of related publish/subscribe projects and the contributions of the PADRES project are presented in Section 1.7, followed by some concluding remarks in Section 1.8.

## 1.2 MESSAGE ROUTING

All interactions in the PADRES distributed content-based publish/subscribe system are performed by routing four messages: advertisements, subscriptions, publications, and notifications. This section outlines the format of each of these messages, then describes how these messages are routed in the PADRES network.

### 1.2.1 Language model

---

<sup>2</sup>The project name PADRES is an acronym that was initially comprised of letters (mostly first letters of first names) of the initial group of researchers working on the project. Over time, the acronym was also synonymously used as name, simply written *Padres*. Both forms are correct. Also, various re-interpretations of the acronym have been published, such as *Publish/subscribe Applied to Distributed REsource Scheduling*, *PAdres is Distributed REsource Scheduling*, etc.

The PADRES language model is based on the traditional [attribute, operator, value] predicates used in several other content-based publish/subscribes systems (Opyrchal, 2000; Carzaniga, 2001; Cugola, 2001; Fabret, 2001; Mühl, 2002; Bittner, 2007). In PADRES, each message consists of a message header and a message body. The header includes a unique message identifier, the message type (publication, advertisement, subscription, or notification), the last and next hops of the message, and a timestamp that records when the message was generated. The content and formats of each message type are detailed below.

## Publications

Data producers, or *publishers*, encapsulate their data in *publication* messages which consist of a comma separated set of [attribute, value] pairs. Each publication message includes a mandatory tuple describing the `class` of the message. The `class` attribute provides a guaranteed selective predicate for matching, similar to the topic in topic-based publish/subscribe systems.<sup>3</sup> A publication that conveys information about a stock listing may look as follows:

```
P: [class, 'STOCK'], [symbol, 'YHOO'], [open, 25.2], [high, 43.0],  
[low, 24.5], [close, 33.0], [volume, 170300], [date, '12-Apr-96']
```

A publication is allowed to traverse the system only if there are data sinks, or *subscribers*, who are interested in the data. Subscribers indicate their interest using subscription messages which are detailed below. If there are no interested subscribers, the publication is dropped. A publication may also contain an optional *payload*, which is a blob of binary data. The payload is delivered to subscribers, but cannot be referenced in a subscription constraint.

## Advertisements

Before a publisher can issue publications, it must supply a template that specifies constraints on the publications it will produce. These templates are expressed via *advertisement* messages. In a sense, an advertisement is analogous to a database schema or a programming language type, and can specify the type and ranges for each attribute as shown in the following example:

```
A: [class, eq4, 'STOCK'], [symbol, isPresent, @STRING], [open, >, 0.0],  
[high, >, 0.0], [low, >, 0.0], [close, >, 0.0], [volume, >, 0],  
[date, isPresent, @DATE]
```

The above advertisement indicates that the publisher will publish only STOCK data with any symbol. The *isPresent* operator allows an attribute to have any value in the domain of the specified type.

An advertisement is said to *induce* publications: the attribute set of an induced publication is a subset of attributes defined in the associated advertisement, and the values of each attribute in an induced publication must satisfy the predicate constraint defined in the advertisement. Note that a publisher may only issue publications that are induced by an advertisement it has sent. Two possible publications P1 and P2 induced by the above advertisement are listed below, while P3 is not induced by the advertisement due to the extra attribute `company`.

```
P1: [class, 'STOCK'], [symbol, 'YHOO'], [open, 25.25],  
[high, 43.00], [low, 24.50]
```

---

<sup>3</sup>The PADRES language is nevertheless fully content-based and supports a rich predicate language.

<sup>4</sup>Operator 'eq' is used for String type values and '=' is used for Integer and float type values.

```
P2: [class, 'STOCK'], [symbol, 'IBM'], [open, 45.25]
```

```
P3: [class, 'STOCK'], [symbol, 'IBM'], [company, 'IBM']
```

## Subscriptions

Subscribers express their interests in receiving publication messages by issuing *subscriptions* which specify predicate constraints on matching publications. PADRES not only allows subscribers to subscribe to individual publications, but also allows correlations or joins across multiple publications. Subscriptions are classified into *atomic* and *composite subscriptions*.

An atomic subscription is a conjunction of predicates. For example, below is a subscription for Yahoo stock quotes.

```
S: [class, eq, 'STOCK'], [symbol, eq, 'YHOO'],  
[open, isPresent, @FLOAT]
```

The commas between predicates indicate the conjunction relation. Similar to publications, each subscription message has a mandatory predicate specifying the `class` of the message, with the remaining predicates specifying constraints on other attributes.

A publication is said to *match* a subscription, if all predicates in the subscription are satisfied by some [attribute, value] pair in the publication. For instance, the above subscription is matched by publications of all YHOO stock quotes with an `open` value. A subscription is said to *cover* another subscription, if and only if any publication that matches the latter also matches the former. That is, the set of publications matching the covering subscription is a superset of those matching the covered subscription.

*Composite subscriptions* consist of atomic subscriptions linked by logical or temporal operators, and can be used to express interest in composite events. A composite subscription is matched only after all component atomic subscriptions are satisfied. For example, the following subscription detects when Yahoo's stock opens at less than 22, and Microsoft's at greater than 31. Parenthesis are used to specify the priority of operators.

```
CS: ([class, eq, 'STOCK'], [symbol, eq, 'YHOO'], [open, <, 22.0]) &&  
([class, eq, 'STOCK'], [symbol, eq, 'MSFT'], [open, >, 31.0])
```

Moreover, unlike the traditional publish/subscribe model, PADRES can deliver not only those publications produced after a subscription has been issued, but also those published before a subscription was issued. That is, PADRES realizes a publish/subscribe model to query both the future and the past (Li, 2007; Li, 2008). In this model, data from the past can be correlated with data from the future. Composite subscriptions that allow correlations across publications continue to work with future data, and also with any combination of historic and future data. In that sense, subscriptions can be classified into *future subscriptions*, *historical subscriptions* and *hybrids* of the two.

For example, the following subscription is satisfied if during the period Aug. 12 to Aug. 24, 2008, MSFT's opening price was lower than the current YHOO opening price. The variable `$X` correlates the opening price in the two stock quotes. This is an example of a hybrid subscription.

```
CS: ([class, eq, 'STOCK'], [symbol, eq, 'YHOO'], [open, eq, $X]) &&  
[class, eq, 'STOCK'], [symbol, eq, 'MSFT'], [open, >, $X],  
[_start_time, eq, '12-Aug-08'], [_end_time, eq, '24-Aug-08']
```

PADRES also provides an SQL-like language called PSQL (PADRES SQL) (Li, 2008), which has the same expressiveness as described above and allows users to uniformly access data produced in the past and future. The PSQL language supports the ability to specify the notification semantic, and it can filter, aggregate, correlate, and project any combination of historic and future data as described below.

In PSQL, subscribers issue SQL-like SELECT statements to query both historic and future publications. Within a SELECT statement, the SELECT clause specifies the set of attributes or aggregation functions to include in the notifications of matching publications, the WHERE clause indicates the predicate constraints to apply to matching publications, and the optional FROM and HAVING clauses help express joins and aggregations.

```
SELECT [ attr | function ], ...  
[FROM src, ...]  
WHERE attr op val, ...  
[HAVING function, ...]
```



The above composite subscription is translated as follows in PSQL.

```
SELECT src1.class, src1.symbol, src1.open, src2.symbol,  
       src2.open  
FROM src1, src2  
WHERE src1.class eq 'STOCK',  
       src2.class eq 'STOCK',  
       src1.symbol eq 'YHOO',  
       src2.symbol eq 'MSFT',  
       src1.open < src2.open,  
       src2.start_time eq '12-Aug-08',  
       src2.end_time eq '24-Aug-08'
```

Notice that the reserved `start_time` and `end_time` attributes can be used to express time constraints in order to query for publications from the past, the future, or both. The sources in the FROM clause specify that two different publications are required to satisfy this query, and are subsequently used to qualify the WHERE constraints. The two publications may come from different publishers and conform to different schema (i.e., advertisements).

The HAVING clause is used to specify constraints across a set of matching publications. The functions  $\text{AVG}(a_i, N)$ ,  $\text{MAX}(a_i, N)$ , and  $\text{MIN}(a_i, N)$  compute the relevant aggregation across attribute  $a_i$  in a window of  $N$  matching publications. The window may either slide over matching publications or be reset when the HAVING constraints are satisfied. The following subscription returns all publications about YHOO stock quotes in a window of 10 publications whose average price exceeds \$20.

```
SELECT class, symbol, price  
WHERE class eq 'STOCK', symbol eq 'YHOO'  
HAVING AVG(price, 10) > 20.00
```

For more information about PSQL, please refer to the technique report (Li, 2008).

## Notifications

When a publication matches a subscription at a broker, a *notification* message is generated and further forwarded into the broker network until delivered to subscribers. Notification semantics do not constrain notification results, but transform them. Recall that notifications may include a subset of attributes in matching publications indicated in the SELECT clause in PSQL. Most existing publish/subscribe systems use matching publication messages as notifications whereas PSQL supports projections and aggregations over matching publications. This simplifies the notifications delivered to subscribers and reduces overhead by eliminating unnecessary information.

### 1.2.2 Broker network and broker architecture

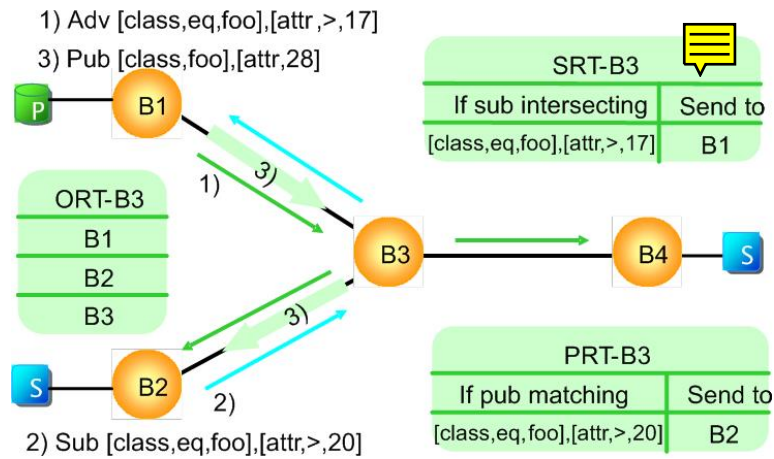


Figure 1. Broker network

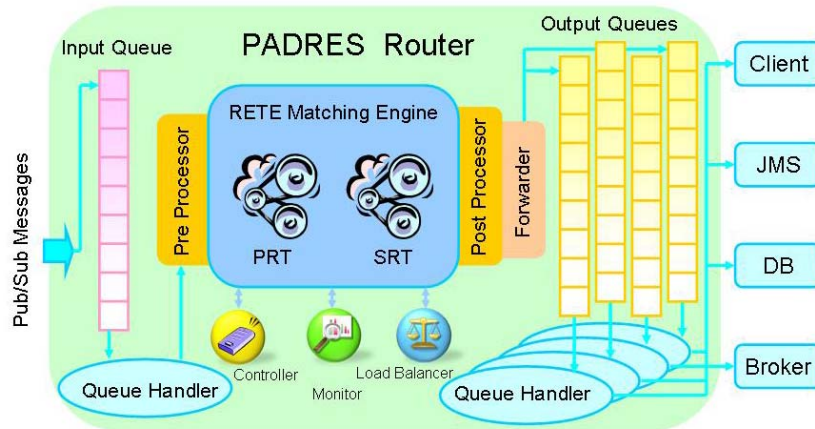


Figure 2. Router architecture

Figure 1 shows a deployed PADRES system consists of a set of brokers connected in an overlay which forms the basis for message routing. Each PADRES broker acts as a content-based router that matches and routes publish/subscribe messages. A broker is only aware of its neighbors (those located within one hop), which information it stores in its *Overlay Routing Tables* (ORT). Clients connect to brokers using various binding interfaces such as Java Remote Method Invocation (RMI) and Java Messaging Service (JMS).

Publishers and subscribers are clients to the overlay. A publisher client must first issue an advertisement before it publishes, and the advertisement is flooded to all brokers in the overlay network. These advertisements are stored at each broker in a Subscription Routing Table (SRT) which is essentially a list of [advertisement, last hop] tuples.

A subscriber may subscribe at any time, and subscriptions are routed based on the information in the SRT. If a subscription intersects an advertisement in the SRT, it is forwarded to the last hop broker the advertisement came from. A subscription is routed hop-by-hop in this way until it reaches the publisher who sent the matching advertisement. Subscriptions are used to construct the Publication Routing Table (PRT). Similar to the SRT, the PRT is a list of [subscription, last hop] tuples, and is used to route publications.

If a publication matches a subscription in the PRT, it is forwarded to the last hop broker of that subscription until it reaches the subscriber that sent the subscription. Figure 1 shows an example PADRES overlay and the SRT and the PRT at one of the brokers. In the figure, in Step 1 an advertisement is published at broker B1. A matching subscription enters through broker B2 in Step 2 and since the subscription overlaps the advertisement at broker B3, it is sent to broker B1. In Step 3 a publication is routed to broker B2 along the path established by the subscription.

Each broker consists of an input queue, a router, and a set of output queues, as shown in Figure 2. A message first goes into the input queue. The router takes the message from the input queue, matches it against existing messages according to the message type, and puts it in the proper output queue(s) which refer to different destination(s). Other components provide other advanced features. For example, the *controller* provides an interface for a system administrator to manipulate a broker (e.g., to shut it down, or to inject a message into it); the *monitor* maintains statistical information about the broker (e.g., the incoming message rate, the average queueing time and the matching time); the *load balancer* triggers offload algorithms to balance the traffic among brokers when a broker becomes overloaded (e.g., the incoming message rate exceeds a certain threshold); and the *failure detector* triggers the fault-tolerance procedure when a failure is detected in order to reconstruct new forwarding paths for messages and ensure timely delivery of publications in the presence of failures.

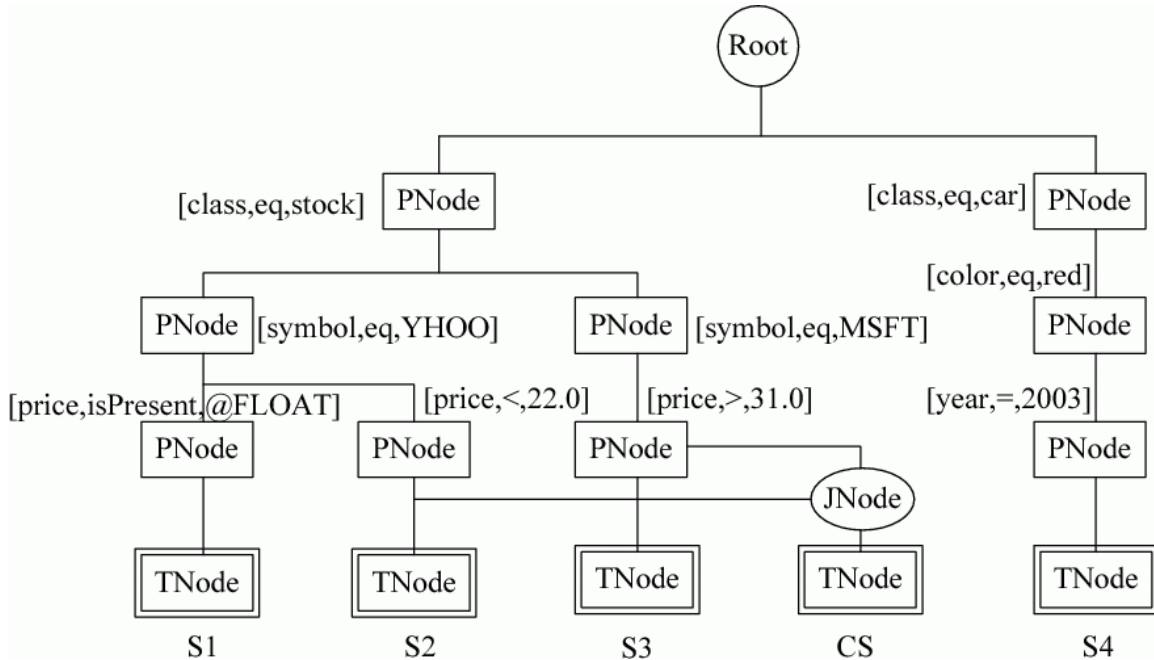


Figure 3. Rete network



PADRES brokers use an efficient Rete-based pattern matching algorithm (Forgy, 1982) to perform publish/subscribe content-based matching. Subscriptions are organized in a Rete network as shown in Figure 3. Each rectangle node in the Rete network corresponds to a predicate and carries out simple conditional tests to match attributes against constant values. Each oval node performs a join between different atomic subscriptions and thus corresponds to composite subscriptions. These oval nodes maintain the partial matching states for composite subscriptions. A path from the root node to a terminal node (a double-lined rectangle) represents a subscription. The Rete matching engine performs efficient content-based matching by reducing or eliminating certain types of redundancy through the use of node sharing. Partial matching states stored in the join nodes allow the matching engine to avoid a complete re-evaluation of all atomic subscriptions each time new publications are inserted into the matching engine. Experiments show that it takes only 4.5 ms to match a publication against 200,000 subscriptions which is nearly 20 times faster than the predicate counting algorithm (Ashayer, 2002). Moreover, the detection time does not increase with the number of subscriptions, but is affected by the number of matched publications. That is, the more publications that match a subscription, the longer it takes the matching engine to process the subscription. This indicates that the Rete approach is suitable for large-scale publish/subscribe systems and can process a large number of publication and subscription messages efficiently. Also, the Rete-based matching engine naturally supports composite subscription evaluation.

### 1.2.3 Content-based routing protocols

Instead of address-based routing, PADRES uses *content-based routing*, where a publication is routed towards the interested subscribers without knowing where subscribers are and how many of them exist. The content-based address of a subscriber is the set of subscriptions issued by the subscriber. This provides a decoupling between the publishers and subscribers.

PADRES provides many content-based routing optimizations to improve efficiency and robustness of message delivery, including covering-based routing, adaptive content-based routing in cyclic overlays, and routing protocols for composite subscriptions.

#### Covering and merging based routing

In content-based publish/subscribe systems, subscribers may issue similar subscriptions. The goal of covering-based routing is to guarantee a compact routing table without information loss, thereby avoiding the propagation of redundant messages, and reducing the size of the routing tables and improving the performance of the matching algorithm.

When a broker receives a new subscription from a neighbor, it performs the following steps to determine how to forward it. First, it searches the routing table to determine if the subscription is covered by some existing subscription from the same neighbor. If it is, the new subscription can be safely removed without inserting it into the routing table and, of course, without forwarding it further. If the new subscription is not covered by any existing subscriptions, the broker checks if it covers any existing subscriptions. If so, the covered subscriptions should be removed.

Subscriptions with no covering relations but which have significant overlap with one another can be *merged* into a new subscription, thus creating even more concise routing tables. There are two kinds of mergers: if the publication set of the merged subscription is exactly equal to the union of the publication sets of the original subscriptions, the merger is said to be *perfect*; otherwise, if the merged subscription's publication set is a superset of the union, it is an *imperfect* merger. Imperfect merging can reduce the number of subscriptions but may allow false positives, that is, publications that match the merged subscription but not any of the original subscriptions. These false positives are eventually filtered out in the network, and subscribers will not receive any false positives, but they do contribute to increased message propagations. However, by selectively and



strategically employing subscription merging the matching efficiency of the publish/subscribe system can be further improved. For additional information, please refer to (Li, 2005; Li, 2008)

### Adaptive content-based routing for general overlays

The standard content-based routing protocol is based on an acyclic broker overlay network. With only one path between any pair of brokers or clients, content-based routing is greatly simplified. However, an acyclic overlay offers limited flexibility to accommodate changing network conditions, is not robust with respect to broker failures, and introduces complexities for supporting other protocols such as failure recovery.

We propose a TID-based content-based routing protocol (Li, 2008) for cyclic overlays to eliminate the above limitations. In the TID-based routing, each advertisement is assigned a unique *tree identifier* (TID) within the broker network. When a broker receives a subscription from a subscriber, the subscription is bound with the TIDs of its matching advertisements. A subscription with a bound TID value only propagates along the corresponding advertisement tree.

Subscriptions set up paths for routing publications. When a broker receives a publication, it is assigned an identifier equal to the TID of its matching advertisement. From this point on, the publication is propagated along the paths set up by matching subscriptions with the same TID without matching the content of the publication at each broker. This is referred to as *fixed publication routing*.

Alternate paths for publication routing are maintained in PRTs as subscription routing paths with different TIDs and destinations. More alternate paths are available if publishers' advertisement spaces overlap or subscribers are interested in similar publications, which is often the case for many applications with long-tailed workloads. Our approach takes advantage of this and uses multiple paths available at the subscription level. Our *dynamic publication routing* (DPR) algorithm takes advantages of these alternate paths by balancing publication traffic among them, and providing more robust message delivery.

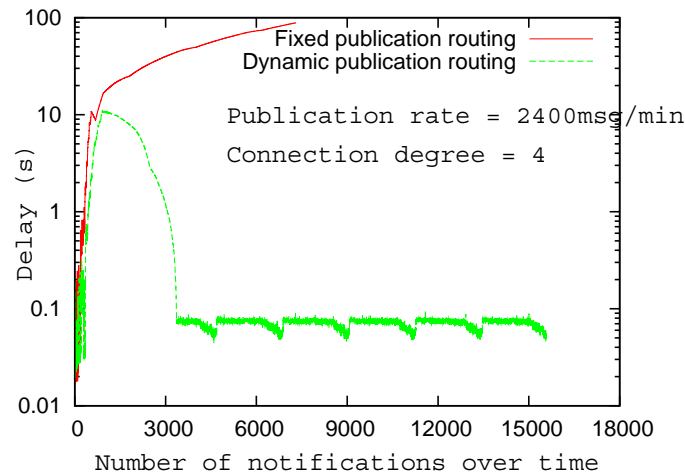


Figure 4. Higher publication rate

We observe in our experiments that an increase in the publication rate causes the fixed routing approach to suffer worse notification delays. For instance, in Figure 4, when the publication rate is increased to 2400 msg/min, the fixed algorithm becomes overloaded with messages queueing up at brokers along the routing path, whereas the dynamic routing algorithm continues to operate by offloading the high workload across alternate paths. The results suggest that dynamic routing is more stable and capable of handling heavier workloads, especially in a well connected network.

## Composite subscription routing

Composite events are detected by the broker network in a distributed manner. In *topology-based composite subscription routing* (Li, 2005), a composite subscription is routed as a unit towards potential publishers until it reaches a broker  $B$  at which the potential data sources are located in different directions in the overlay network. The composite subscription is then split at broker  $B$ , which is called the *join point broker*. Each component subscription is routed to potential publishers separately. Later, matching publications are routed back to the join point broker for it to detect the composite event. Notice that topology-based routing assumes an acyclic overlay and does not consider dynamic network conditions.

In a general (cyclic) broker overlay, multiple paths exist between subscribers and publishers, and topology-based composite subscription routing does not necessarily result in the most efficient use of network resources. For example, composite event detection would be less costly if the detection is close to publishers with a higher publishing rate, and in a cyclic overlay, more alternative locations for composite event detection may be available. The overall savings are significant if the imbalance in detecting composite events at different locations is large. PADRES includes a *dynamic composite subscription routing* (DCSR) algorithm (Li, 2008) that selects optimal join point brokers to minimize the network traffic and matching delay while correctly detecting composite events in a cyclic broker overlay. The DCSR algorithm determines how a composite subscription should be split and routed based on the cost model discussed below.

A broker routing a composite subscription makes local optimal decisions based on the knowledge available to itself and its neighbors. The cost function captures the use of resources such as memory, CPU, and communication. Suppose a composite subscription  $CS$  is split at broker  $B$ . The *total routing cost* (TRC) of  $CS$  is:

$$TRC_B(CS) = RC_B(CS) + \sum_{i=1}^n RC_{B_{N_i}}(CS_{B_{N_i}})$$

and includes the *routing cost* of  $CS$  at broker  $B$ , denoted as  $RC_B(CS)$ , and those neighbors where publications contributing to  $CS$  may come from, denoted as  $RC_{B_{N_i}}(CS_{B_{N_i}})$ .  $CS_{B_{N_i}}$  denotes the part of  $CS$  routed to broker  $B_{N_i}$ , and may be an atomic or composite subscription.

The cost of a composite subscription  $CS$  at a broker includes not only the time needed to match publications (from  $n$  neighbors) against  $CS$ , but also the time these publications spend in the input queue of the broker, and the time that matching results (to  $m$  neighbors) spend in the output queues. This cost is modeled as

$$RC_B(CS) = \sum_{i=1}^n T_{in} |P(CS_{B_{N_i}})| + \sum_{i=1}^n T_m |P(CS_{B_{N_i}})| + \sum_{i=1}^m T_{out_i} |P(CS)|$$

where  $T_m$  is the average matching time at a broker,  $T_{in}$  and  $T_{out_i}$  are the average time messages spend in the input queue, and output queue to the  $i^{th}$  neighbor.  $|P(S)|$  is the cardinality of subscription  $S$ , which is the number of matching publications per unit time. To compute the cost at a neighbor, brokers periodically exchange information such as  $T_{in}$  and  $T_m$ . This information is incorporated into an M/M/1 queueing model to estimate queueing times at neighbor brokers as a result of the additional traffic attracted by splitting a composite subscription there.

Evaluations of the DCSR algorithm were conducted on the PlanetLab wide-area network with a 30 broker topology. The metrics measured include the bandwidth of certain brokers located on the composite subscription routing path. In Figure 5, the solid bars represent the number of outgoing messages at a broker, and the hatched bars are the number of incoming messages that are not forwarded. Note that the sum of the solid and hatched bars represents the total number of incoming messages at a broker. Three routing algorithms are compared: simple routing, in which

composite subscriptions are split into atomic parts at the first broker, topology-based composite subscription routing, and the *DCSR* algorithm. The topology-based routing imposes less traffic than simple routing by moving the join point into the network and the *DCSR* algorithm further reduces traffic by moving the join point closer towards congested publishers as indicated by the cost model. In the scenario in Figure 5, compared to simple routing, the *DCSR* algorithm reduces the traffic at Brokers *B1* by 79.5%, a reduction that is also enjoyed by all brokers downstream of the join point.

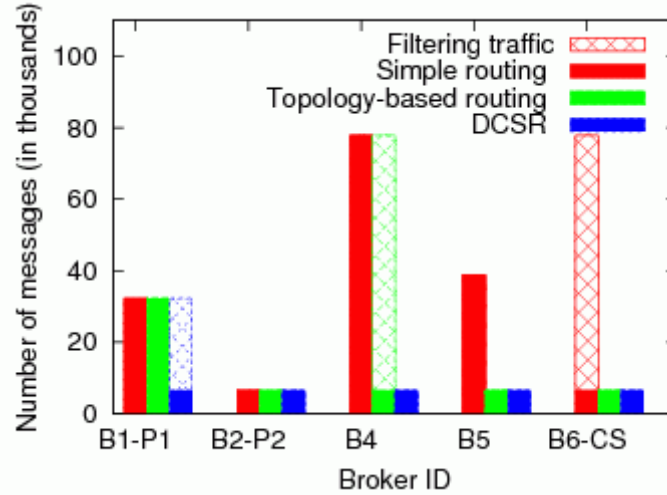


Figure 5. Composite subscription traffic

### 1.3 Historic Data Access

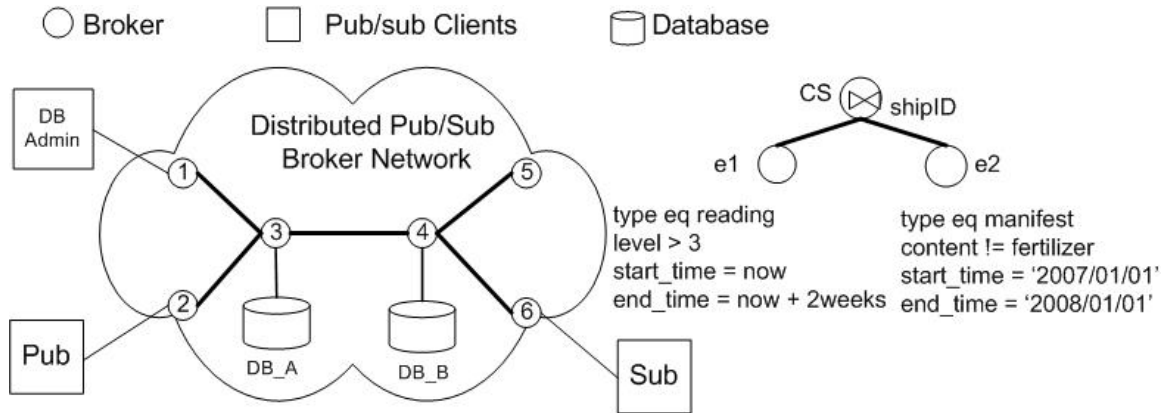


Figure 6. Historic data access architecture

PADRES allows subscribers to access both future and historic data with a single interface as described in Section 1.2.1. The system architecture, shown in Figure 6, consists of a traditional distributed publish/subscribe overlay network of brokers and clients. Subscriptions for future publications are routed and handled as usual (Opyrchal, 2000; Carzaniga, 2001). To support historic subscriptions, databases are attached to a subset of brokers as shown in Figure 6. The databases

are provisioned to sink a specified subset of publications, and to later respond to queries. The set of possible publications, as determined by the advertisements in the system, is partitioned and these partitions assigned to the databases. A partition may be assigned to multiple databases to achieve replication, and multiple partitions may be assigned to the same database if database consolidation is desired. Partition assignments can be modified at any time, and replicas will synchronize among themselves. The only constraint is that each partition be assigned to at least one database so no publications are lost. Partitioning algorithms as well as partition selection and assignment policies are described in (Li, 2008). Subscriptions can be atomic expressing constraints on single publications or composite expressing correlation constraints over multiple publications. We describe their routing under the extended publish/subscribe model.

### 1.3.1 Atomic Subscription Routing

When a broker receives an atomic subscription, it checks the `start_time` and `end_time` attributes. A future subscription is forwarded to potential publishers using standard publish/subscribe routing (Opyrchal, 2000; Carzaniga, 2001). A hybrid subscription is split into future and historic parts, with the historic subscription routed to potential databases as described next.

For historic subscriptions, a broker determines the set of advertisements that overlap the subscription, and for each partition, selects the database with the minimum routing delay. The subscription is forwarded to only one database per partition to avoid duplicate results. When a database receives a historic subscription, it evaluates it as a database query, and publishes the results as publications to be routed back to the subscriber. Upon receiving an `END` publication after the final result, the subscriber's host broker unsubscribes the historic subscription. This broker also unsubscribes future subscriptions whose `end_time` has expired.

### 1.3.2 Adaptive Routing

Topology-based composite subscription routing (Li, 2005) evaluates correlation constraints in the network where the paths from the publishers to subscriber merge. If a composite subscription correlates a historic data source and a publisher, where the former produces more publications, correlation detection would save network traffic if moved closer to the database, thereby filtering potentially unnecessary historic publications earlier in the network. Based on this observation, the *DCSR* algorithm we discussed in Section 1.2.3 can be applied here. The `WHERE` clause constraints of a composite subscription can be represented as a tree where the internal nodes are operators, leaf nodes are atomic subscriptions, and the root node represents the composite subscription. A composite subscription example is represented as the tree in Figure 6. The recursive *DCSR* algorithm (Li, 2008) computes the destination of each node in the tree to determine how to split and route the subscription. The algorithm traverses the tree as follows: if the root of the tree is a leaf, that is, an atomic subscription, the atomic subscription's next hop is assigned to the root. Otherwise, the algorithm processes the left and right children's destination trees separately. If the two children have the same destination, the root node is assigned this destination, and the composite subscription is routed to the next hop as a whole. If the children have different destinations, the algorithm estimates the total routing cost for potential candidate brokers, and the mini-

mum cost destination is assigned to the root. If the root's destination is the current broker, the composite subscription is split here, and the current broker is the join point and performs the composite detection. The algorithm assigns destinations to the tree nodes bottom up.

When network conditions change, join points may no longer be optimal and should be recomputed. A join point broker periodically evaluates the cost model, and upon finding a broker able to perform detection cheaper than itself, initiates a join point movement. The state transfer from the original join point to the new one includes routing path information and partial matching states. Each part of the composite subscription should be routed to the proper destinations so routing information is consistent. Publications that partially match composite subscriptions stored at the join point broker must be delivered to the new join point.

For more detailed description of the historic data access function, please refer to our technique report (Li, 2008).

## 1.4 Load Balancing

In a distributed publish/subscribe system, geographically dispersed brokers may suffer from uneven load distributions due to different population densities, interests, and usage patterns of end-users. A typical scenario is an enterprise-scale deployment consisting of a dozen brokers located at different world-wide branches of an international corporation, where the broker network provides a communication service for hundreds of publishers and thousands of subscribers. It is conceivable that the concentration of business operations and departments, and thus publish/subscribe clients and messages, is orders of magnitudes higher at the corporate headquarters than at the subsidiary locations. Such hotspots at the headquarters can overload the broker there in two ways. First, the broker can be overloaded, if the incoming message rate into the broker exceeds the maximum processing or matching rate of the broker's matching engine. Because the matching rate is inversely proportional to the number of subscriptions in the matching engine, this effect is exacerbated if the number of subscribers is large (Fabret, 2001). Second, overload can also occur if the output transmission rate exceeds the total available output bandwidth. In both cases, input queues at the broker accumulate with messages waiting to be processed, resulting in increasingly higher processing and delivery delays. Worse yet, the broker may crash when it runs out of memory from queueing too many messages.

The matching rate and both the incoming and outgoing message rates determine the load of a broker. In turn, these factors depend on the number and nature of subscriptions that the broker services. Thus, load balancing is possible by offloading *specific* subscribers from higher loaded to lesser loaded brokers. The PADRES system supports this capability using load estimation methodologies, a load balancing framework, and three offload algorithms (Cheung, 2006).

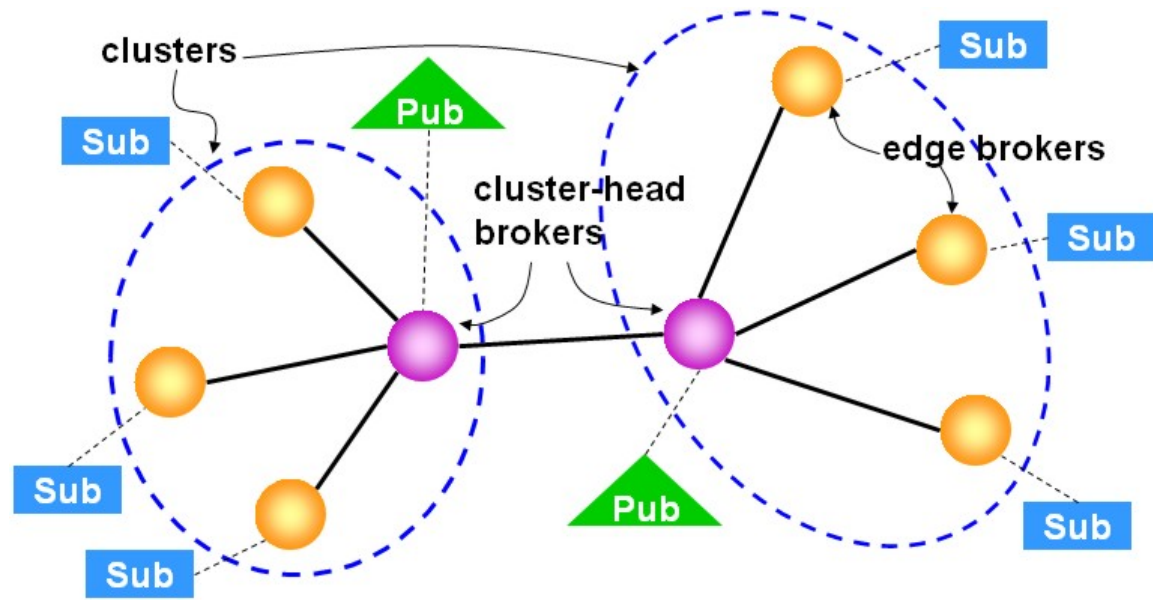


Figure 7. PEER architecture

The load balancing framework consists of the *PADRES Efficient Event Routing* (PEER) architecture, a distributed load exchange protocol called *PADRES Information Exchange* (PIE), and detection and mediation mechanisms at the local and global load balancing tiers. The PEER architecture organizes brokers into a hierarchical structure as shown in Figure 7. Brokers with more than one neighboring broker are referred to as *cluster-head brokers*, while brokers with only one neighbor are referred to as *edge brokers*. A cluster-head broker with its connected set of edge brokers, if any, forms a *cluster*. Publishers are serviced by cluster-head brokers, while subscribers

are serviced by edge brokers. Load balancing is possible by moving subscribers among edge brokers of the same or different cluster. With PIE, edge brokers within a cluster exchange load information by publishing and subscribing to PIE messages of a certain cluster ID. For example, a subscription to PIE messages from cluster C01 is `[class, eq, 'LOCAL_PIE']`, `[cluster, eq, 'C01']`. The detector invokes load balancing if it detects overload or the load of the local broker is greater than another broker by a threshold. Load is characterized by three load metrics. First, the input utilization ratio ( $I_r$ ) captures the broker's input load and is calculated as:

$$I_r = \frac{i_r}{m_r}$$

where  $i_r$  is the rate of incoming publications and  $m_r$  is the maximum message match rate calculated by taking the inverse of the matching delay. Second, the output utilization ratio captures the output load and is calculated as:

$$O_r = \left( \frac{t_{busy}}{t_{window}} \right) \left( \frac{b_{rx}}{b_{tx}} \right)$$

where  $t_{window}$  is the monitoring time window,  $t_{busy}$  is the amount of time spent sending messages within  $t_{window}$ ,  $b_{rx}$  represents the messages (in bytes) put into the output queue in time window  $t_{window}$ , and  $b_{tx}$  represents the messages (in bytes) removed from the output queue and sent successfully in time window  $t_{window}$ . A utilization value greater than 1.0 indicates overload. Third, the matching delay captures the average amount of time to match a publication message.

The core of the load estimation is the PADRES *Real-time Event to Subscription Spectrum* (PRESS), which uses an efficient bit vector approach to estimate the input and output publication loads of all subscriptions at the local broker. Together with *locally subscribing* to the load-accepting broker's *covering subscription set*, PRESS can estimate the amount of input and output load introduced at the load-accepting broker for all subscriptions at the offloading broker.

Each of the three offload algorithms are designed to load balance on each load metric of the broker by selecting the appropriate subscribers to offload based on their profiled load characteristics. Simultaneously, the subscriptions that each offload algorithm picks minimize the impact on the other load metrics to avoid instability. For example, the match offload algorithm offloads subscriptions with the minimal traffic, and the output offload algorithm first offloads highest traffic subscriptions that are covered by the load accepting broker's subscription(s.)

This solution inherits all of the most desirable properties that make a load balancing algorithm flexible. PIE contributes to the *distributed* and *dynamic* nature of the load balancing solution by allowing each broker to invoke load balancing whenever necessary. *Adaptiveness* is provided by the three offload algorithms that load balance on a unique performance metric. The local mediator gives *transparency* to the subscribers throughout the offload process. Finally, load estimation with PRESS allows the offload algorithms to account for broker and subscription *heterogeneity*.





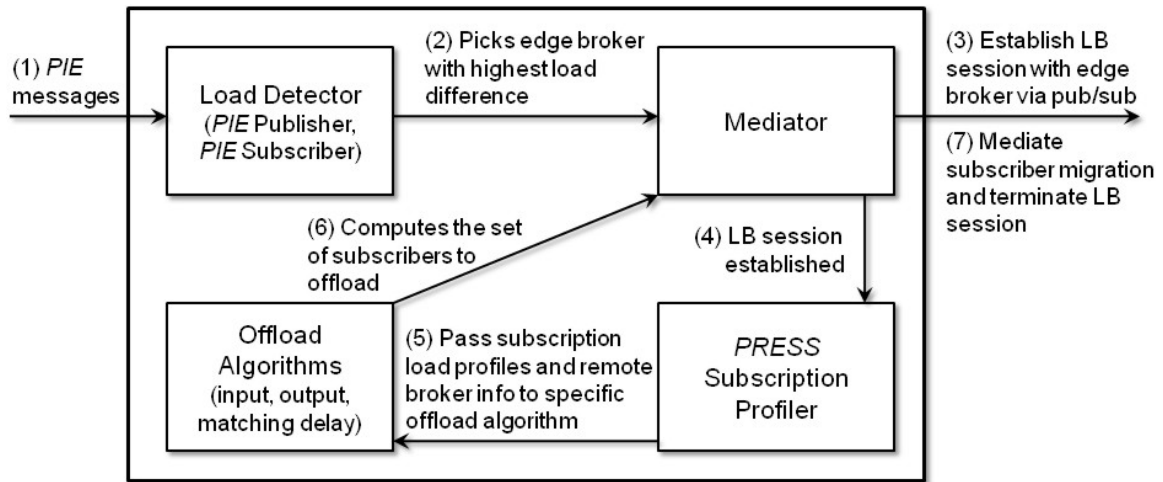


Figure 8. Components of the load balancer

The components that make up the load balancing solution, shown in Figure 8, consist of the detector, mediator, load estimation tools, and offload algorithms. The detector detects and initiates a trigger when an overload or load imbalance occurs. The trigger from the detector tells the mediator to establish a load balancing session between the *offloading broker* (broker with the higher load doing the offloading) and the *load-accepting broker* (broker accepting load from the offloading broker). Depending on which performance metric is to be balanced, one of the offload algorithms is invoked on the offloading broker to determine the set of subscribers to delegate to the load-accepting broker based on estimating how much load is reduced and increased at each broker using the load estimation algorithms. Finally, the mediator is invoked to coordinate the migration of subscribers and ends the load balancing session between the two brokers. The load balancing solution is integrated as a stand-alone module in the PADRES broker as shown in Figure 9.

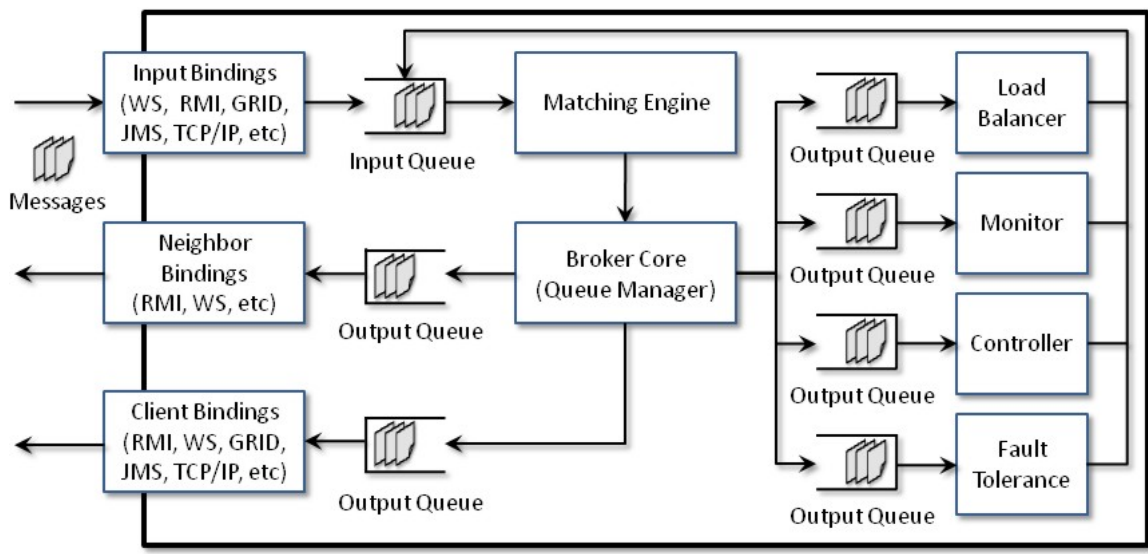


Figure 9. Internal view of the PADRES broker

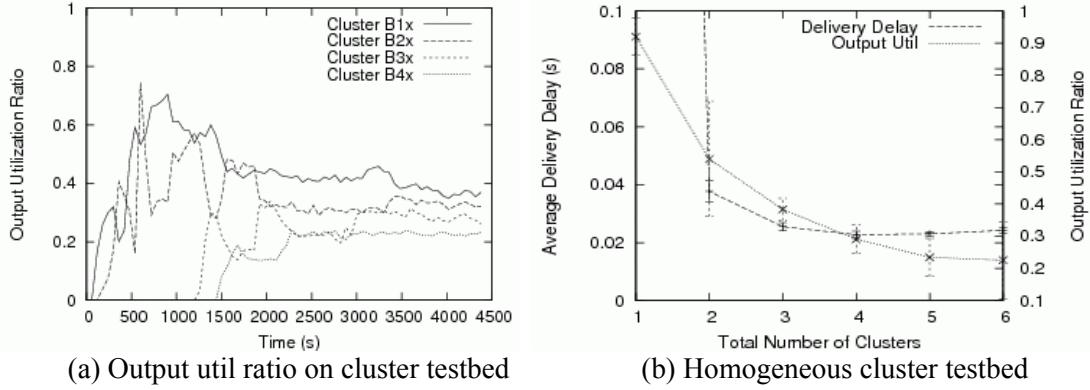


Figure 10. Experiment results

Evaluations of the PADRES load balancing algorithms in the shared wide-area PlanetLab testbed, a dedicated local cluster environment, and a simulator all show that the load balancing algorithms prevent overload by distributing subscribers while simultaneously balancing the three load metrics among edge brokers. The algorithms are effective in both homogeneous and heterogeneous environments and enable the system to scale with added resources. Figure 10(a) shows an experiment with four heterogeneous clusters arranged in a chain with two edge brokers per cluster and having all subscribers join at an edge broker on one end of the chain, namely B1x. As time progresses, subscribers get distributed to other clusters down the chain until the algorithm converges around 3500 s into the experiment. Not shown on this graph is the observation that the subscriptions that sink higher traffic are assigned to brokers with more computing capacity than to brokers with limited capacity. Figure 10(b) shows that the average load of the brokers decreases as more resources (in the form of clusters) are added. Simultaneously, delivery delay decreases when going from two to four clusters, but increases beyond five clusters due to a longer path length. By adaptively subscribing to load information, the message overhead of the load balancing infrastructure is only 0.2% in the experiments on the cluster testbed. The results also show that a naive load balancing solution that cannot identify subscription space and load are not only inefficient but can also lead to system instability. The interested reader may consult the full paper for more details about the algorithms and the experiments (Cheung, 2008).

## 1.5 Fault-tolerance

Fault-tolerance in general refers to the ability of a system to handle the failure of its components and maintain the desired quality of service (QoS) under such conditions. Furthermore, a  $\delta$ -fault-tolerant system operates correctly in presence of up to  $\delta$  failures. A common class of failures in a distributed system is node *crashes*, in which nodes stop executing instructions, no longer send or receive messages, and lose their internal state. Failures may be *transient* in which case nodes may *recover* by executing a recovery procedure.

In order to achieve  $\delta$ -fault-tolerance, PADRES nodes (brokers and clients) transparently collect additional routing information as part of the normal operation of the system and use this information to react to the failure of their neighbors. Two types of information are collected: the broker topology, and the subscription routes. The former allows for increased network connectivity and prevents partitions forming as a result of failures, and the latter is used to decide among alternative routing paths and avoid interruptions in publication delivery.

The remainder of this section presents the system-wide *consistency* properties that correspond to the routing topology and subscription routing state, and describes how this information is used to achieve fault-tolerant routing and recovery.

### 1.5.1 Consistency

To ensure correct operation of the system (in presence of up to  $\delta$  failure) the topology and subscription routing information must be kept *consistent* at all times. In our context, consistency is dependant on the desired degree of fault-tolerance of the system,  $\delta$ , and is thus referred to as  $\delta$ -consistency. The value of  $\delta$  is chosen by an administrator based on a number of factors including the fan-out of brokers, rate of failures, and average downtimes. To achieve  $\delta$ -consistency for topology routing information, brokers must know about all peers within a  $(\delta+1)$ -neighborhood. Distances are measured over the initial acyclic topology which acts as a backbone for the entire system. The  $\delta$ -consistent topology routing information enhances the connectivity of this acyclic structure by enabling brokers to identify and connect to not only their neighbors, but all nodes within distance  $\delta+1$ .

On the other hand,  $\delta$ -consistency for subscription routing information is achieved by maintaining references to certain brokers along the subscription propagation paths. These references point to brokers that are up to  $\delta+1$  hops closer to the subscriber. More specifically, a broker that is within distance  $\delta+1$  of a subscriber stores the subscriber's broker ID as the reference. If it is farther, then the reference points to another broker along the path to the subscriber. This broker is  $\delta+1$  hops closer to the subscriber. Figure 11 illustrates a sample network with  $\delta$ -consistent subscription routing information for two highlighted subscribers.

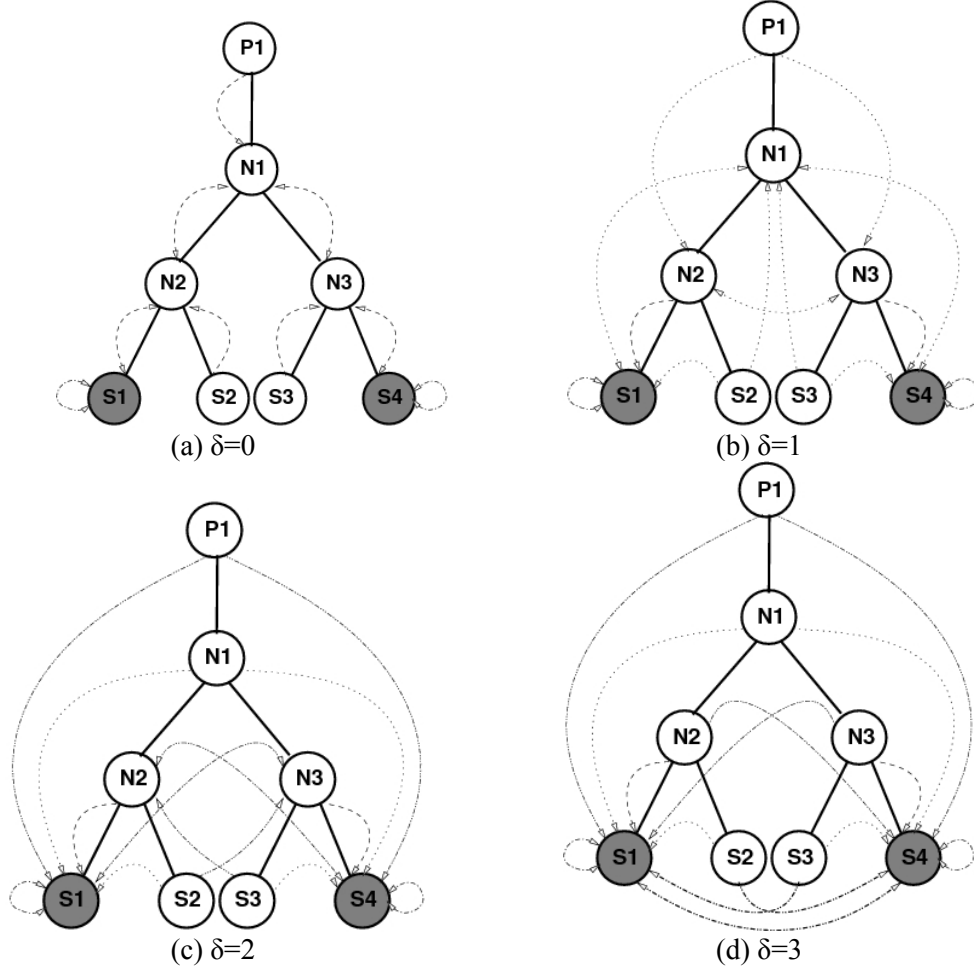


Figure 11.  $\delta$ -consistent subscription routing information for two subscribers S1 and S2. An arrow from A to B indicates that A holds a reference pointer to B.

### 1.5.2 Fault-Tolerant Forwarding Algorithm

When there are no failures in the system, brokers are connected to their immediate neighbors in the acyclic backbone topology. At the same time, they continuously monitor their communicating peers using a heartbeat based failure detector. It is assumed that the failure detector works perfectly and all broker failures are detected after some time. When a failure is detected, the fault-tolerant forwarding algorithm is triggered at the non-faulty neighbors.

The main objective of fault-tolerant forwarding is to bypass failed neighboring brokers and re-establish the publication flows. For this purpose, having detected a failure, brokers create new communication links to the immediate neighbors of their failed peer, as illustrated in Figure 12. These new brokers, identified using the local topology routing information, may themselves concurrently try to bypass the failed node. Endpoint brokers that establish a new connection (to bypass a failed node) perform an initial handshake and exchange their operational states. Additionally, they exchange the sequence number of the last message tagged by the other endpoint (or “null” if there is no such message). This information is used to determine whether messages previously sent to the failed broker need to be retransmitted. Subsequently, nodes start to (re-)send outstanding messages over a new link if a matching subscriber is reachable through the link. This process maintains the initial arrival order of messages and uses subscription reference pointers to decide to which new peers to forward the messages.

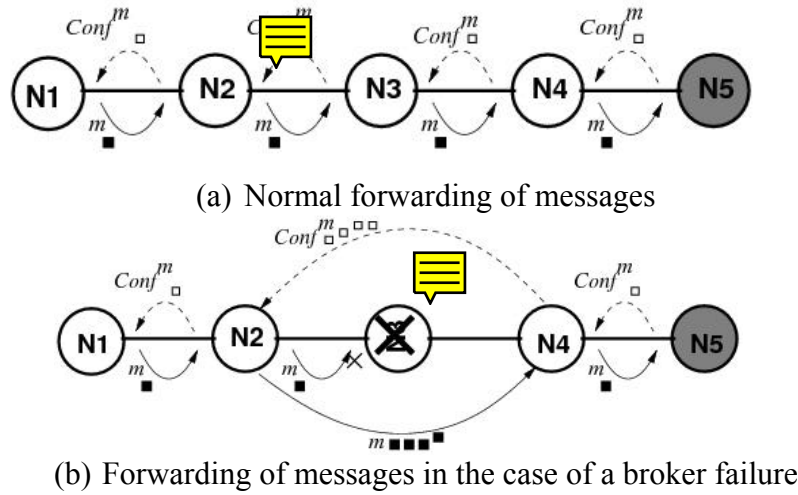


Figure 12. Fault-tolerance forwarding bypasses faulty brokers (filled circles represent final destinations).

### 1.5.3 Recovery Procedure

The *recovery procedure* is executed by brokers that have experienced failures in the past and enables them to re-enter the system and participate in message routing. This is in contrast to the fault-tolerance forwarding algorithm that runs on non-faulty brokers, in order to deliver messages in presence of failed brokers. The *recovering* brokers have lost their internal state due to the failure, and have further missed messages (e.g., subscriptions) that were sent during their downtime. Thus, the main objective of the recovery procedure is to restore this lost internal state by establishing a  $\delta$ -consistent topology and subscription routing information. The recovery procedure involves the following steps: (i) identify previous location in the topology, and the nearby brokers;

(ii) synchronize and receive routing information; (iii) participate in message forwarding; (iv) end recovery and notify peers.

Recovering brokers can identify their previous location in the topology by accessing their local persistent storage or querying a discovery service that maintains this data. In either case, it is necessary that prior to failures brokers persistently store their topology routing information to disk, or properly update the discovery service about changes to the topology. The synchronization step involves connecting to the closest non-faulty brokers and requesting updated routing information. Reference pointers in the received subscription routing information is properly manipulated such that the  $\delta$ -consistency requirements are met.

The synchronization step may involve several nearby brokers and may be lengthened as large volumes of data are transferred or as new failures occur. During this period, new subscription messages may be inserted into the system and the topology tree may undergo further changes. To enable the recovering brokers to keep up with this updated information, they participate in message forwarding in a similar way to fully operational peers. The only exception is that the synchronization points attach additional information determining the destinations of the messages. This is required since the routing information of a recovering broker at this stage may not be complete. Once all the recovery information is transferred from all synchronization points, the recovery is complete and the peers are appropriately notified. From this point onward, the  $\delta$ -consistent routing properties are established and the recovered broker fully participates in regular message forwarding. More information about the fault-tolerance and recovery procedure is provided in (Sherafat, 2007; Sherafat, 2008).

## 1.6 Tools

PADRES includes a number of tools to help manage and administer a large publish/subscribe network. This section presents two of these tools: a monitor that allows a user to visualize and interact with brokers in real time, and a deployment tool that simplifies the provisioning of large broker networks.

### 1.6.1 Monitor

The PADRES monitor lets a user monitor and control a broker federation. It is implemented as a regular publish/subscribe client and performs all its operations using the standard publish/subscribe interface and messages. Among other benefits, this allows the monitor to be run from anywhere a connection to a broker can be established, and to access any broker in the federation including those that would otherwise be hidden behind a firewall.

Once connected to a broker, the monitor issues a subscription for broker status information that is periodically published by all brokers in the system. This information is used to construct a visual representation of brokers, overlay links, and clients. The display is updated in real time as the monitor continuously discovers and receives updates from brokers.

Figure 13 shows a screenshot of the PADRES monitor connected to a federation of 100 brokers. Nodes in the visualization may be rearranged by manually dragging the nodes around, or various built-in graphing algorithms can decide on the layout automatically. Detailed information of each broker, such as the routing tables, system properties, and various performance metrics can also be viewed. In terms of the control features, the user can pause, resume, and shutdown individual brokers; inject any type of message (including advertisement, unadvertisement, subscription, unsubscription, and publication messages) at any point in the network; and trace and visualize the propagation paths of messages. For more information, please refer to PADRES user guide (Jacobsen, 2004).

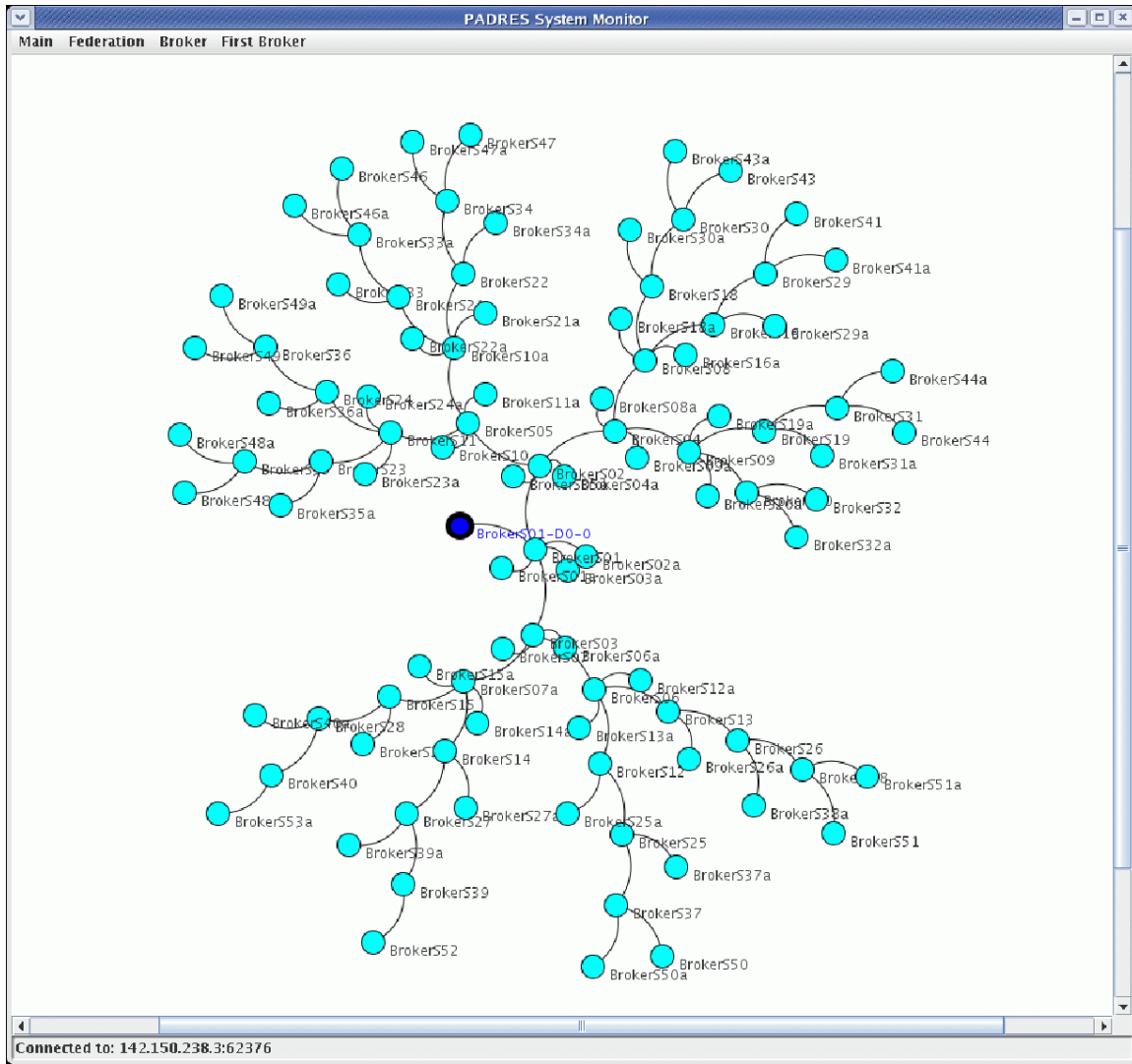


Figure 13. PADRES monitor showing 100 brokers

### 1.6.2 PANDA

The *PADRES Automated Node Deployer and Administrator* (PANDA) simplifies the installation, deployment, and management of large broker networks distributed among any number of machines. In addition to starting and terminating processes, PANDA can install and uninstall the required Linux RPM packages, upload and remove PADRES binaries, and even retrieve broker log files from remote machines. As all remote operations are executed via SSH commands, PANDA can manage the deployment of brokers on any machine with SSH access. In addition, PANDA is fully compatible with the PlanetLab wide-area research testbed (PlanetLab, 2006).

PANDA lets a user easily describe complex broker and client networks in a flexible *topology file*. For example, the following ADD command in a topology file indicates that a broker process named *BrokerA* is to be started immediately upon deployment on machine 10.0.1.1 with a set of custom properties, such as a port number of 10000 and an ID of *Alice*.

```
0.0 ADD BrokerA 10.0.1.1 startbroker.sh -Xms 64 -Xmx 128 -hostname
10.0.1.1 -p 10000 -i Alice
```



That same process can be scheduled to be terminated at a particular time, say 3500sec, with a REMOVE command as follows:

```
3500 REMOVE BrokerA 10.0.1.1
```

All IP addresses referenced in the topology file will be taken into consideration when an install or upload command is issued to install RPMs or upload tarballs. PANDA also supports a unique 2-phase deployment scheme for brokers. Phase 1 includes all broker processes with deployment time of 0.0 where the user wishes the brokers and its overlay links to be fully up and running before deploying any other processes in Phase 2. Phase 2 includes all those processes with deployment times greater than 0. Phase 2 starts only when Phase 1 is complete as detected by PANDA's built-in monitor. For more details and examples of PANDA's topology file, please refer the online PADRES user guide (Jacobsen, 2004).

The PANDA architecture, shown in Figure 14, consists of a Java program that uses helper shell scripts to interact with the remote nodes. The user interacts with PANDA through a text console. Upon loading a topology file or entering a command directly into the console, the input is parsed for correctness and UNIX commands are generated by the CommandGenerator. A TopologyValidator validates the input, checking for errors such as duplicate broker IDs. When the user enters the deploy command, the DeploymentCoordinator orchestrates the 2-phase deployment and executes remote UNIX command operations through the ScriptExecutor.

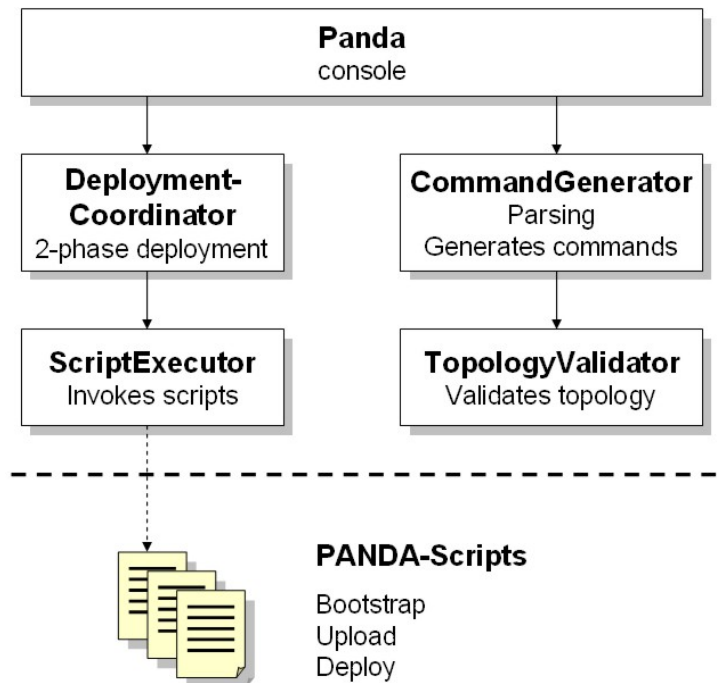


Figure 14. PANDA architecture

These features of PANDA greatly simplify the management of large broker networks. They can also be used to fully automate any PADRES experiments including starting and stopping brokers and clients at certain times and collecting the experiment log files.

## 1.7 Applications



The simple yet powerful publish/subscribe interface supported by PADRES can be applied to a variety of scenarios that run the gamut from simple consumer news filtering to complex enterprise applications.

What follows in this section are illustrations of how the design of sophisticated applications can be simplified by capitalizing on the various features of the publish/subscribe middleware outlined in the preceding sections. Some of these applications exploit properties of the publish/subscribe *model* itself such as the expressiveness of the publish/subscribe language that enables fine-grained event filtering, event correlation and context-awareness, the complex interaction patterns that can be realized such as many to many conversations, the natural decoupling of components that allows for asynchronous and anonymous communication, and the push-based messaging that enables applications to react to events in real-time. Certain scenarios below also take advantage of features of the PADRES publish/subscribe *middleware* including the scalability achieved by a distributed broker architecture, the ability to dynamically load-balance the brokers, and fault-tolerance capabilities that enable the brokers to automatically detect and recover from failures.

To convey the breadth of scenarios to which publish/subscribe can be applied, applications from three domains are presented: *consumer* applications used by individuals for personal productivity or entertainment purposes, *enterprise* applications that are critical to the operation of a business entity, and *infrastructure* services that are used to deploy, monitor, and manage hardware and software infrastructures. These three domains, of course, may overlap and are not necessarily mutually exclusive.

### 1.7.1 Consumer applications

Interactions in consumer or end-user applications generally follow a client-server model in which a user receives service from a service provider, or a peer-to-peer (P2P) model where a group of users interact in an ad-hoc manner.

The client-server interaction model is very simple to achieve in publish/subscribe systems: the server, such as a newscast service, simply publishes its data, and the clients subscribe to the subset of data they are interested in. Of course, the usability of this type of application depends on the expressiveness with which the clients can express their interests, making an expressive content-based language, such as that provided by PADRES, preferable over simpler publish/subscribe models.

The PADRES content-based publish/subscribe system can also be used to construct a P2P information dissemination service thanks to its distributed broker overlay and its capability to route messages in cyclic networks (Li, 2008). The two primary concerns in P2P-type application are handling node churning and reducing message overhead in information dissemination. As the earlier sections of this chapter have shown, the PADRES distributed broker overlay and content-based routing can efficiently address these issues.

### News Services

Conventional Internet-based newscast services may use a topic-based publish/subscribe system where news items are published under certain topics (for example, “sports” or “local-news”) and customers subscribe to one or more of these topics. When the customers want to fine-tune their subscriptions, the topics have to be sub-divided to match their interests. This is handled by creating a hierarchy of topics. For example, when a user wants to subscribe to Canadian Olympic events, a topic hierarchy of “sports → Olympics → Canada” is created as shown in Figure 15.a.

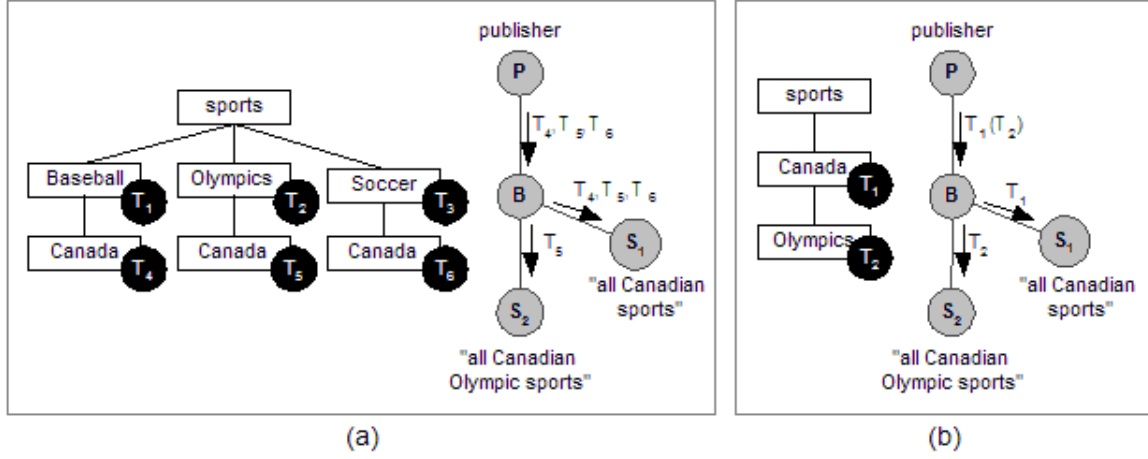


Figure 15: Topic hierarchy and subscription covering.

The major issue with a topic hierarchy is its limited expressiveness. Users are constrained to subscribe to only the topics defined by the hierarchy, but constructing a hierarchy to cover all the potential combinations of user interests leads to an explosion of topics and results in poor matching performance and management overhead. Furthermore, when the user interests change, either the topic hierarchy must be restructured, or users must subscribe to broader topics than they are interested. The first solution is impractical, and the second solution results in redundant message overhead and requires additional processing by the clients to filter out publications that are not of interest.

A content-based publish/subscribe system avoids these issues because client interests are expressed using fine-grained attribute-value tuples. Table 1 and Figure 15 show the difference between topic-based and content-based systems in expressing subscriber interests. Note that the topic-based system can match **Sub 2** exactly with topic  $T_5$ , but there is no topic that exactly matches **Sub 3**. Therefore, the client is forced to subscribe to the superset topic  $T_2$  which covers **Sub 3** but also contains unrelated news items. On the other hand, as shown in the table, the results can be filtered more accurately in content-based system by including all the necessary attribute-value tuples in the subscription.

Subscription	Topic-based System (Figure 15)	PADRES Content-based Language
<b>Sub 1</b> : “all Canadian sports news”	$T_4, T_5, T_6$	<code>[class, eq, 'sports'], [country, eq, 'Canada']</code>
<b>Sub 2</b> : “all Canadian Olympic news”	$T_5$ (exact match)	<code>[class, eq, 'sports'], [country, eq, 'Canada'], [event, eq, 'Olympics']</code>
<b>Sub 3</b> : “all 2008 Olympic news”	$T_2$ (super set)	<code>[class, eq, 'sports'], [event, eq, 'Olympics'], [year, eq, '2008']</code>

Table 1. Subscribing using topic-based and content-based publish/subscribe systems.

The topic hierarchy also influences the distribution of the matching workload in a distributed system. Consider the network shown in Figure 15 where subscribers  $S_1$  and  $S_2$  connect to broker **B** and subscribe to **Sub 1** and **Sub 2**, respectively. When the topic hierarchy is constructed as shown in Figure 15.a, the publisher matches **Sub 1** to topics  $T_4, T_5, T_6$  and **Sub 2** to topic  $T_5$  and forwards the news items on these topics to broker **B** which forwards them to the respective subscribers. When the topic hierarchy is organized as in Figure 15.b, however, broker **B** need only subscribe to topic  $T_1$ , because both **Sub 1** and **Sub 2** are covered by this topic. When the broker

receives the news items on  $T_1$ , it can immediately forward them to  $S_1$ , and forward those news items that match  $T_2$  to  $S_2$ . In this way, the matching workload is distributed in the system making the system more scalable.

It is difficult, however, to design a topic hierarchy that effectively distributes the matching workload while simultaneously offering topics that closely correspond to all user interests. This issue does not arise in content-based systems because the language model provides a way of covering subscriptions as described in Section 1.2.1. For example, Table 1 shows that the content-based definition of **Sub 1** covers that of **Sub 2**. Therefore, a content-based publish/subscribe system provides a more scalable design.

## Intelligent Vehicular Ad-Hoc Networks

A more sophisticated application of the content-based publish/subscribe paradigm is an intelligent vehicular ad-hoc networks (InVANET). Present day smart car functions involve making decisions based on the data fed from different sensors embedded within a car's infrastructure, such as accident prevention using a proximity radar or air-bag deployment using deceleration sensor.

Automobiles in an InVANET collaborate with one another to construct a distributed sensor that captures the collective knowledge of the individual sensors in each vehicle. The information dissemination in InVANET follows a reactive model where a car detecting a situation triggers an action from another car. A content-based system like PADRES can efficiently implement this event-driven architecture, with each car playing the role of a publisher, subscriber, and content-based router. In this scenario, the events of interest will include accidents, traffic jams, or even the events of cars leaving parking spots.

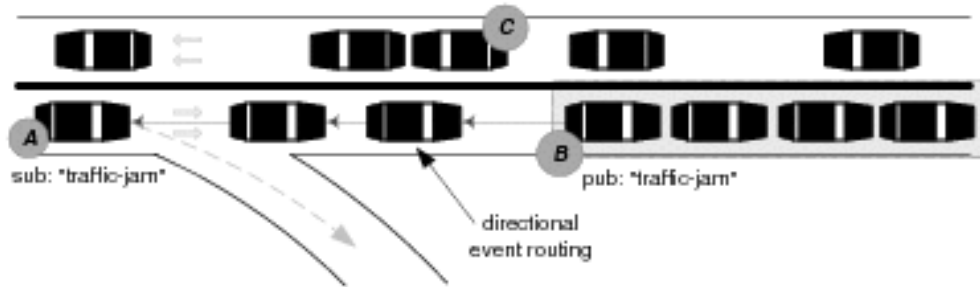


Figure 16. An example scenario in a InVANET system.

An example scenario is illustrated in Figure 16. Car *A* is interested in knowing about traffic jams in advance so that it can take an alternate path. It subscribes to a “traffic-jam” event as:

```
[class, eq, 'traffic-jam'], [location, <, MY_LOC + 10], [dir, eq, 'HW401W']
```

Note that the subscription includes location and directional (HW401W, i.e., Highway 401, West bound) constraints. The location variable `MY_LOC` is substituted with the current GPS coordinates. The subscription is propagated in the overlay created by the smartcars. When Car *B* detects a traffic jam (perhaps using its internal sensors), it publishes a “traffic-jam” event as:

```
[class, 'traffic-jam'], [location, MY_LOC], [dir, 'HW401W']
```

This event is reverse-propagated through the overlay until it reaches Car *A*.

A similar publish/subscribe scheme can be used to find a newly available parking spot: when a car leaves a parking spot in a busy downtown area, it can publish the event which is propagated to cars whose driver is interested in finding a parking spot in the vicinity.

An InVANET requires a publish/subscribe middleware that can be implemented over ad-hoc cyclic networks. It should also be noted that subscriptions and publications include location and directional attributes which should be exploited during event routing to reduce message overhead. For example, in Figure 10, the “traffic-jam” event generated by Car *B* need not reach Car *C* that is traveling in the opposite direction. PADRES provides the necessary infrastructure to construct an InVANET, and its matching engine can be extended to support directional and location operators in its subscription language.

### 1.7.2 Enterprise applications

The number of applications and users an enterprise manages and supports as well as the amount of data that flows between them grows larger with a growing enterprise. Therefore, enterprises enforce automated service management infrastructures to scale with a growing service base. This management infrastructure automates the detection of application states; it automates the triggering of certain activities based on the detected application states; and it orchestrates the interaction between different applications and users. These activities require *complex event processing* (CEP) that accepts the different application states as events and process them to detect certain *situations* and invoke relevant actions. A content-based publish/subscribe system, especially a distributed system like PADRES, is the ideal choice for implementing a CEP infrastructure. The applications and users can join the system as clients, situations can be defined by composite event subscriptions, and the interactions between the applications are managed by subscribing to certain events (application states or situations).

### Sensor Networks

Sensor networks are created by interconnecting a number of sensors monitoring different parameters at different locations. Sensor networks are commonly used in environment monitoring, traffic control, health care, and battlefield surveillance. Event processing is the primary operation in a sensor network and a content-based publish/subscribe system can simplify this operation. Each sensor can be considered as a publisher that outputs a constant stream of data with a fixed schema (advertisement). The applications that process the sensor data can subscribe to various events from different sensors and produce their own events. For example, a tsunami event can be detected using a composite subscription:

```
([class, eq, 'seismic'], [magnitude, >, 3], [location, =, $L], [time, =, $T]) &&
([class, eq, 'wave'], [height, >, 10], [location, <, $L + 5], [time, <, $T + 10])
```

It detects an event of a seismic activity of magnitude larger than 3 followed by (within 10min) a wave with a height of more than 10m at a location within 5km from the origin of the seismic event. When this condition is satisfied, a new alert event can be produced as:

```
[class, 'climate-alert'], [condition, 'tsunami'], [location, $L], [time, $T]
```

where the values for \$L and \$T are extracted from the publications that triggered the detection of the subscription given above.

A *radio frequency identification* (RFID) system is a type of sensor network that has already been successfully used in monitoring moving objects. Tracking books in a library, inventory of goods in a store, and automated payments in toll highways are few of the applications of RFID-enabled tags. At present, RFID tags are used mostly to identify the presence (or the lack of presence) of an item at a certain location at a given time. If the RFID readers are networked, the time stamped detection events can be conveyed as publications to a content-based publish/subscribe system that will increase the functionality of the RFID-based systems. For example, a shoplifting event can be detected by subscribing to an appropriate composite event: detecting an event with a certain RFID at the exit sensor without detecting it at a sales counter.

In a sensor network, the event schemata are mostly constant and simple, but the sensors are distributed and the amount and rate of data produced by them (publications) are often very large. This requires a distributed, fast, and scalable matching and routing infrastructure like PADRES (Petrovic, 2005). In addition, the publish/subscribe middleware used in sensor networks should be self-configuring and fault tolerant, because the sensor networks are sometimes implemented on a mobile network where the lifetimes and the locations of the nodes vary constantly.

## Business Process Management

*Business process management* (BPM) is another important business application where content-based publish/subscribe systems are extremely useful. Business process management organizes a set of enterprise applications and processes in order to facilitate efficient communication among themselves and with clients. One of the key aspects of BPM is *workflow processing*. Figure 17 shows an example workflow of an online retailer.

A workflow describes the interactions between different enterprise applications, processes, and users and includes causal and temporal relationships between applications. Because it follows the model of an event-driven system where the completion of one or many processes activates another, a content-based system can be used to implement it. For example, Figure 17 shows the hypothetical workflow of an online sales application where the availability of an item should be checked and the shipping charge should be calculated before enabling a detailed item view. Therefore, the 'ItemView' module should subscribe to a composite event:

```
([class, eq, 'INVOKE'], [service, eq, 'ItemView'], [id, eq, $X]) &&  
([class, eq, 'RESULT'], [service, eq, 'AvailCheck'], [id, eq, $X]) &&  
([class, eq, 'RESULT'], [service, eq, 'CalcShipping'], [id, eq, $X])
```

Note that this subscription performs two tasks: the first part of the composite subscription provides the activation command to the 'ItemView' module, but the other two parts restrict the module to be activated only after the results from the relevant 'AvailCheck' and 'CalcShipping' services are received. These events are connected using a variable on the *id* attribute.

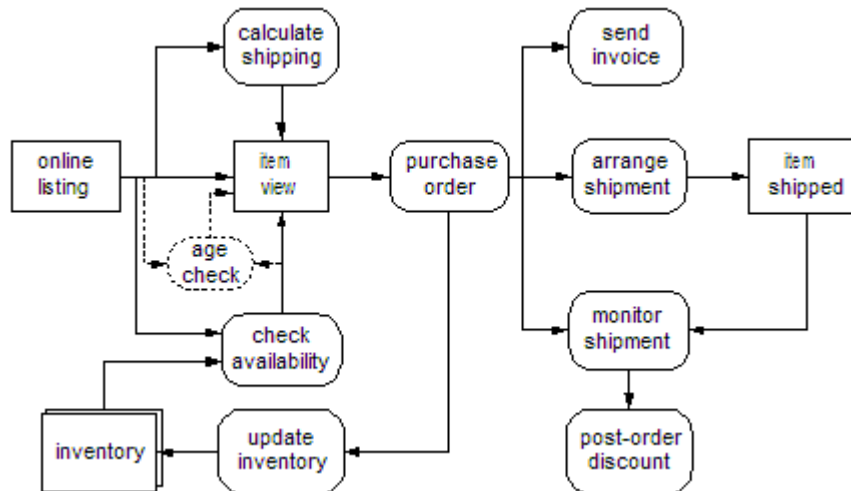


Figure 17. Example workflow of an online sales application.

In a workflow, the output of a process can vary depending on the incoming event parameters. For example, in Figure 17, the online retailer might decide to give a post-order discount, if the shipment is delayed more than a specific duration. When a purchase order is placed, a shipment monitor is instantiated as well, which waits for the event of shipment. When the shipment event is received, it will check the purchase agreement and if the shipment failed to match the agreed shipment date, a post-order discount is sent out. This operation can be performed by issuing the following subscription and publication:

```

Sub: ([class, eq, 'INVOKE'], [service, eq, 'MonitorShipment'], [id, eq, $X],
      [time, =, $T]) &&
      ([class, eq, 'RESULT'], [service, eq, 'ItemShipped'], [id, eq, $X],
      [time, >, $T + 10])
Pub: [class, 'INVOKE'], [service, 'PostOrderDiscount'], [id, 'aaaa']

```

The subscription is used to detect the condition where a post-order discount is to be issued and the publication is used to activate the post-order delivery module. The threshold that triggers a post order discount is 10 days from the order date.

A content-based publish/subscribe system not only efficiently implements a workflow, but it also simplifies reorganizing the workflow when required. For example, in Figure 17, the retailer may decide to invoke a new service to verify the age of the consumers against the approved limit before activating the item view. The modification, shown with the dotted lines in the figure, can be readily implemented by unsubscribing the previous subscription and invoking a new subscription as follows:

```

([class, eq, 'INVOKE'], [service, eq, 'ItemView'], [id, eq, $X]) &&
([class, eq, 'RESULT'], [service, eq, 'AvailCheck'], [id, eq, $X]) &&
([class, eq, 'RESULT'], [service, eq, 'CalcShipping'], [id, eq, $X]) &&
([class, eq, 'RESULT'], [service, eq, 'AgeCheck'], [id, eq, $X],
[approve, eq, 'YES'])

```

## Business Application Monitoring

*Business application monitoring* (BAM) is another aspect of BPM. It involves continuously monitoring the performance of applications or processes, producing reports, and triggering actions or notifications when some specific conditions are met.

Again, BAM concerns can be easily realized with publish/subscribe middleware by adding components that subscribe to events generated by various distributed application monitor software agents. Event processing can be used to detect various system conditions and take actions accordingly. The loose coupling properties of publish/subscribe are exploited here to allow BAM components to monitor applications without having to instrument the application they are monitoring. Rather, they simply subscribe to events they are interested in.

## Enterprise Service Bus

*Service-oriented architectures* (SOA) have become a common solution for the problems of enterprise application management. In an SOA, applications are constructed by composing a set of reusable services that are available through standardized interfaces. These applications are themselves exposed as yet another service that can in turn be composed by other applications. An *enterprise service bus* (ESB) plays a central role in an SOA, mediating the interactions among the services. A publish/subscribe middleware such as PADRES supports the core functionality required of an ESB, and is ideally suited to serve as an ESB in an SOA.

Another important component of an SOA is a *service registry* where services are registered and discovered. In publish/subscribe terminology, the services can be presented as both publishers and subscribers. The historic query capabilities of PADRES can be used to discover services that have registered in the past, while the usual publish/subscribe subscription mechanisms enable applications to continuously monitor for and be notified of new services. The actual execution of a workflow of composed services can then be achieved as described in the BPM discussion above.



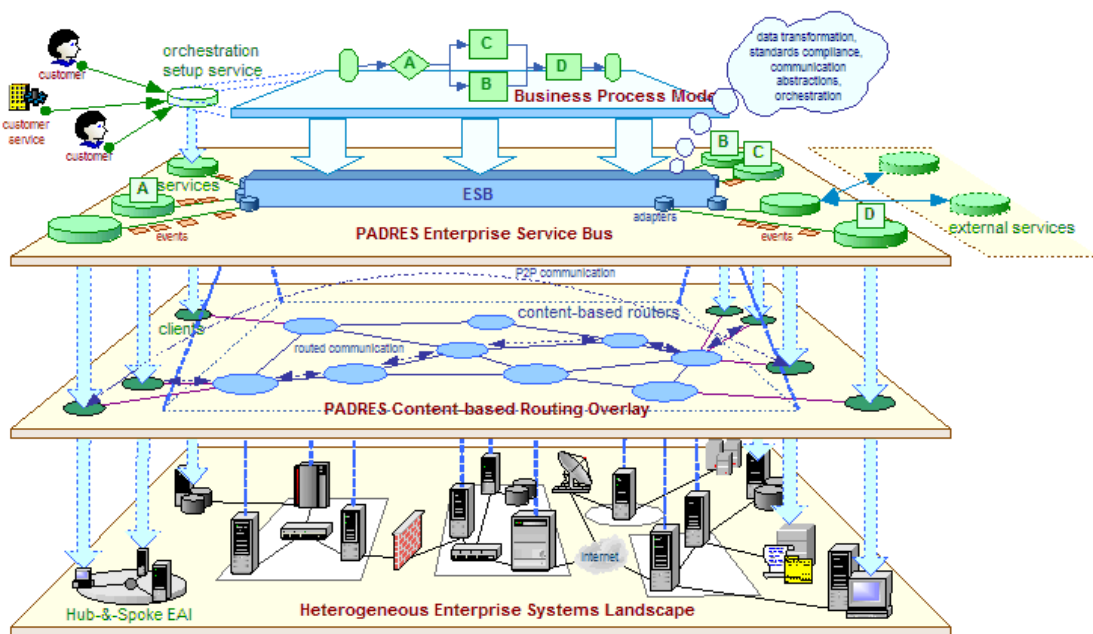


Figure 18. Building an enterprise service bus with PADRES.

Two of the key responsibilities of an ESB are service orchestration and governance. A business orchestration defines the interaction among the services (creating workflow descriptions), whereas governance concerns the management of corporate policies regarding the hosted services and their interactions. An ideal ESB should support an event-driven distributed architecture, an expressive workflow description, a standard-based integration model, flexible data transformation capabilities, and an autonomous and federated environment.

The PADRES middleware provides a highly distributed event-driven architecture that supports many of the required ESB characteristics. Figure 18 shows how PADRES can be used to implement an ESB. The distributed broker network connects applications across a large enterprise, even across the Internet, and the content-based publish/subscribe system provides an event-driven system that supports expressive workflows. Service orchestration is supported by the content-based resource discovery and event-based monitoring and the situation detection mechanism helps enforce governance policies. In addition, the fault tolerant and load balancing capabilities of PADRES provide the means to create a robust ESB.

### 1.7.3 Infrastructure applications

In an era where applications are increasingly hosted on a distributed infrastructure, composed of components from various partners, used by unknown and possibly hostile users, and yet demand reliable performance, an application developer's concerns no longer end with the development and testing of the product. Effectively monitoring and managing the applications and infrastructure at run-time is necessary to provide an acceptable level of service to users. Owing to its clean decoupling, expressive filtering capabilities and matching predication ability (Liu, 2009), the content-based publish/subscribe paradigm is ideally suited for realizing complex real-time management solutions (Yan, 2009).

## Intrusion detection

One set of infrastructure concerns relates to monitoring a system for malicious attacks. Typically, administrators will specify rules or signatures of attacks they wish to monitor. An attack signature on a Web server may be an excessive request rate for web pages from the same IP address, whereas a credit card fraud may be detected if the same card is used from multiple locations within a short period of time.

For example, the following composite subscription will detect any user attempting to probe a machine for certain open ports.

```
([class, eq, 'TCPOPEN'], [dest_port, =, 21], [src_ip, eq, $X],  
[dest_ip, eq, $Y]) &&  
([class, eq, 'TCPOPEN'], [dest_port, =, 22], [src_ip, eq, $X],  
[dest_ip, eq, $Y]) &&  
([class, eq, 'TCPOPEN'], [dest_port, =, 110], [src_ip, eq, $X],  
[dest_ip, eq, $Y])
```

To monitor for this signature, a client can issue the above subscription at any time while the system is operational. Likewise, removing the rule is a simple unsubscribe operation by the client and has no effect on the network. The composite subscription above performs in-network filtering so the monitoring client is only notified if the rule is matched, and distributed composite subscription matching algorithms ensure that the rule is detected at the optimal point in the network. For example, if one particular machine in the network is the target of many TCP sessions, the composite subscription detection can automatically move closer to that machine so that events do not have to propagate far into the network before the pattern is matched.

There may be a concern that it is not efficient to require network components to publish events for every conceivable operation such as TCP session state, or link utilization. However, the PADRES system is designed so that events that are of no interest to anyone in the system are immediately dropped and incur no overhead further in the network. For example, if the above subscription is unsubscribed, then TCPOPEN publication would be dropped (assuming there are no other subscriptions interested in these events).

The real power of monitoring a system using the publish/subscribe paradigm is seen when monitoring rules span multiple layers in a system. For example an attack signature may require monitoring both the network infrastructure and application behavior. For example, an attempt to break into a machine and use it as a spam relay may be defined by a pattern of a number of unsuccessful login attempts, followed by a successful one, after which a number of emails are sent.

```
([class, eq, 'LOGIN'], [userid, eq, $U], [status, eq, 'FAIL'],  
[src_ip, eq, $S]) &&  
([class, eq, 'LOGIN'], [userid, eq, $U], [status, eq, 'FAIL'],  
[src_ip, eq, $S]) &&  
([class, eq, 'LOGIN'], [userid, eq, $U], [status, eq, 'SUCCESS'],  
[src_ip, eq, $S], [dest_ip, eq, $D]) &&  
( [class, eq, 'TCPOPEN'], [dest_port, =, 25], [src_ip, eq, $D],  
[dest_ip, eq, $X]) &&  
([class, eq, 'TCPOPEN'], [dest_port, =, 25], [src_ip, eq, $D],  
[dest_ip, eq, $Y])
```

In the above subscription, the LOGIN publications are generated by the authentication server (at the application layer), and the network layer components issue the TCPOPEN publications, but the subscription nevertheless is able to retrieve and correlate them in a uniform manner.

Another advantage of using the publish/subscribe model to perform intrusion detection is that the attack signatures are monitored in real-time as the system is running, instead of analyzing log

files after the fact. This is important in situations, such as credit card fraud detection where the attack must be managed as soon as possible to prevent further damage.

The ability to correlate events from different layers or applications in a diverse system is also useful in diagnosing the cause of a problem. For example, in an environment where a set of services are deployed on a cluster of machines, an administrator may wish to know which services were invoked shortly before a machine becomes overloaded.

```
([class, eq, 'INVOKE'], [service, eq, $A], [time, >, $T - 10s]) &&  
([class, eq, 'CPULOAD'], [machine_id, eq, 'OVERLOAD'], [time, =, $T])
```

The above subscription would return all invocations up to 10 seconds before the overload event, and an administrator can use this information to isolate the potential cause of the overload.

## Service level agreements

System administrators need to be concerned not only with malicious attacks, but with legitimate usage from end users or partners that may affect their applications in unexpected or undesirable ways. To monitor whether their applications are providing (and receiving) the desired performance, it is common for businesses to define service level agreements (SLAs) on large-scale enterprise applications. An SLA defines a contract between a service provider and consumer and precisely outlines how the consumer will use the service, and states the guarantees offered by the provider. Often penalties of not abiding to the terms are also specified in the SLA.

Consider an online retailer running a sales business process shown in Figure 17. Some of the services in this process, such as the shipping services, may be outsourced to other businesses. To provide an acceptable performance to their users, however, the retailer may require certain service guarantees from the shipping services. For example, the retailer may demand that the *Calculate Shipping* service in Figure 17 respond to requests within 0.5 seconds 99.9% of the time within any 24 hour window, and a failure to meet this level of service will result in a loss of payment for that window. Conversely, the SLA may also specify that the retailer will not invoke the *Calculate Shipping* service more than 100 times per minute within any one minute window.

Monitoring of such SLAs can be implemented cleanly using a publish/subscribe model (Chau, 2008; Muthusamy, 2008). Suppose the process in Figure 17 is executed in a distributed PADRES execution engine. In this case, invocations of and results from services are represented by publications. The loose coupling of the publish/subscribe paradigm allows the SLA monitoring subsystem to subscribe to these publications without altering the process execution or even stopping the running process. Subscriptions to retrieve the invocation and result publications may look as follows.

```
Sub1: [class, eq, 'INVOKE'], [service, eq, 'CalcShipping']  
Sub2: [class, eq, 'RESULT'], [service, eq, 'CalcShipping']
```

The monitoring subsystem issuing the above subscription would have to correlate the invocation and result publications and compute the time difference between the two. However, with more complex correlation and aggregation capabilities, it is possible to issue a single subscription that calculates the time difference, and returns the result to the subscriber. For example, the following composite subscription will correlate invocations with their appropriate results and return pairs of publications to the monitoring client. This saves the monitoring client from having to perform the correlation and allows the correlation processing to occur in the network at the optimal point.

```
([class, eq, 'INVOKE'], [service, eq, 'CalcShipping'], [id, eq, $X]) &&  
([class, eq, 'RESULT'], [service, eq, 'CalcShipping'], [id, eq, $X])
```

In general, SLAs can be modeled as three types of components: metrics, service level objectives (SLOs), and actions.<sup>5</sup> Metrics measure some phenomenon such as the time when a product is delivered, or the number of times an item is purchased. Sometimes a distinction is made between atomic metrics which measure a property directly, and composite metrics that aggregate the measurements from other metrics. For example, the occurrence of late shipments may be measured by an atomic metric, and the total number of late shipments per day response computed by a composite metric. SLOs are a Boolean expression of some desired state. For example, a desired threshold on the number of late shipments per day may be denoted as *LateShipmentsPerDay* < 10. Finally, actions are descriptions of what should occur when an SLO is violated. Examples of actions include sending an email to a sales manager, generating a publication (that will be processed by another component), and writing to a log file.

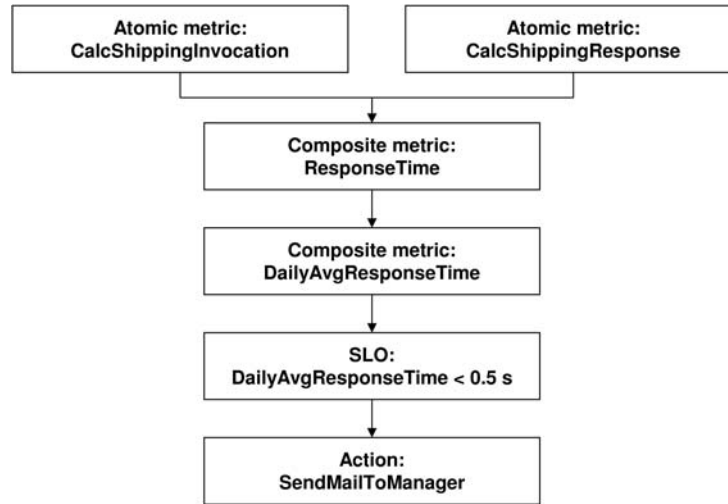


Figure 19. SLA monitoring components

Each metric, SLO, and action in an SLA can be mapped to a publish/subscribe client. For example, consider an SLA where an email should be sent to the sales manager if the average daily response time of the *Calculate Shipping* service is greater than 0.5 seconds. Figure 19 shows metrics, SLOs, and actions that realize this SLA. Each component in the figure is modeled as publish/subscribe client, and interactions between them are through publish/subscribe messages. These clients together realize the monitoring of the SLA.

The distributed monitoring architecture coupled with a distributed publish/subscribe system scales well to large SLAs or those that are processing a high volume of publications. Notice that the clients utilize the filtering and in-network aggregation provided by the content-based publish/subscribe model, and being publish/subscribe clients, automatically benefit from the dynamic load-balance and fault-tolerance properties of the system.

An example of in-network filtering is that only those events from the *DailyAvgResponseTime* client in Figure 19 that report a response time greater than 0.5 seconds are delivered to the SLO client. The remaining events are dropped and incur no overhead in the remainder of the network. The *ResponseTime* metric in Figure 19 issues a composite subscription for correlated response and invocation events of the *Calculate Shipping* service. This composite subscription aggregates these pairs of events in the network such that, for example, invocation events with no corresponding response events are not propagated. Furthermore, the load balancing algorithms will

<sup>5</sup>These terms are borrowed from the Web Service Level Agreements (WSLA) specification.

ensure that the placement of these clients does not result in load imbalances that may affect the monitoring of the SLA or the execution of the process it is monitoring.

SLAs need not be limited to ensuring that services provided by partners perform as expected, but may also be used internally by an enterprise to monitor various business measures such as the length of time users spend browsing before purchasing an item online, the average time spent on a tech support call, or the number of products that are returned within a month. All these monitoring tasks can be efficiently performed on existing applications without modifying or restarting them.

## 1.8 Related work

While publish/subscribe was first implemented in centralized client-server systems, current research focuses mainly on distributed versions. The key benefit of distributed publish/subscribe is the natural decoupling of publishers and subscribers. Since the publishers are unconcerned with the potential consumers of their data, and the subscribers are unconcerned with the locations of the potential producers of interesting data, the client interface of the publish/subscribe system is powerful yet simple and intuitive.

**Language model:** There are several different classes of publish/subscribe systems. *Topic-based* publish/subscribe (Oki, 1993) has a topic associated with each publication which indicates the region of interest for contained data. Clients subscribing to a particular topic would receive all publications with the indicated topic. Topics are similar to the notion of *groups* used in the context of *group communication* (Powell, 1996). *Content-based* publish/subscribe systems add significant functionality by allowing subscribers to specify constraints on the data within a publication. In contrast to the *topic-based* approach, publications are classified according to their content. SIENA (Carzanig, 2001), REBECA (Mühl, 2002), Gryphon (Opyrchal, 2000), Le Subscribe (Fabret, 2001), and ToPSS (Liu, 2002; Liu, 2004) are some well-known content-based publish/subscribe prototypes.

**Content-based routing:** Distributed Content-based publish/subscribe systems typically utilize *content-based routing* in lieu of the standard address-based routing. Messages in content-based routing are routed from source to destination based entirely on the content of the messages. Since publishers and subscribers are decoupled, a publication is routed towards the interested clients without knowing specifically where those clients are and how many such clients exist. Effectively, the content-based *address* of a subscriber is the set of subscriptions it has issued. Two versions of content-based routing are known: simple routing, for example Gryphon (Opyrchal, 2000), and covering-based routing which is discussed in SIENA (Carzanig, 2001), subscription summarization (Triantafillou, 2004) and JEDI (Cugola, 2001). Merging-based routing (Mühl, 2002) is an advanced version of covering based routing. PADRES (Li, 2005; Li, 2008) extends merging-based routing with imperfect merging capabilities that can offer further performance benefits.

**General overlays:** Most publish/subscribe systems assume an acyclic overlay network. For example, REBECA (Fiege, 2002) explores advanced content-based routing algorithms based on an acyclic broker overlay network, and JEDI (Cugola, 2001) uses a hierarchical overlay for event dispatching. SIENA (Carzaniga, 2001), however, proposes a routing protocol for general overlay networks using reverse path forwarding to detect and discard duplicate messages. In SIENA, any advertisement, subscription or publication message may be duplicated. As well, routing path adaptations to changing network conditions and the implications for composite event detection are not addressed. PADRES (Li, 2008) explores the alternative paths available in a general overlay to provide adaptive and robust message delivery in content-based publish/subscribe systems.

There have been attempts to build content-based publish/subscribe systems over group multicasting primitives such as IP multicast (Deering, 1990). To appreciate the challenge in doing so, consider a scenario with  $N$  subscribers. In a content-based system, each message may be deliv-

ered to any subset of these subscribers, resulting in  $2^N$  “groups”. It is infeasible to manage such exponentially increasing numbers of groups, and the algorithms seek to construct a limited number of groups such that the number of groups any given message must be sent to is minimized and the precision of each group is maximized (i.e., minimize the number of group members that are not interested in events sent to that group). This is an NP-complete problem (Adler, 2001), but there have been attempts to develop heuristics to construct such groups (Opyrchal, 2000; Riabov, 2002). To avoid these complexities more recent content-based routing algorithms (Carzanig, 2001; Cugola, 2001) have abandoned the notion of groups and rely on an overlay topology that performs filtering and routing based on message content.

There have been a number of content-based publish/subscribe systems that exploit the properties of distributed hash tables (DHT) to achieve reliability. Hermes (Pietzuch, 2002) builds an acyclic routing overlay over the underlying DHT topology but does not consider alternate publication routing paths as in PADRES. Other approaches (Gupta, 2004; Muthusamy, 2005; Aekaterinidis, 2006; Muthusamy, 2007) construct distributed indexes to perform publish/subscribe matching. PastryStrings (Aekaterinidis, 2006) is a comprehensive infrastructure for supporting rich queries with range and comparison predicates on both numerical and string attributes. It can be applied in a publish/subscribe environment with a broker network implemented using a DHT network. The distinguishing feature of PastryStrings is that it shows how to leverage specific DHT infrastructures to ensure logarithmic message complexity for both publication and subscription processing. Meghdoot (Gupta, 2004) is a content-based publish/subscribe system build over the CAN DHT. For an application with  $k$  attributes, Meghdoot constructs a CAN space of dimension  $2k$ . Subscriptions are mapped to a point in the CAN space and stored at the responsible node. Publications traverse all regions with possible matching subscriptions. Meghdoot handles routing load by splitting a subscription at a peer to its neighbors. P2P-ToPSS (Muthusamy, 2005), unlike other DHT publish/subscribe systems which focus on large-scale benefits, focuses on small-scale networks. It shows that in small networks (with less than 30 peers) DHTs continue to exhibit good storage load balance of (key,value) pairs, and lookup costs. PADRES, on the other hand, assumes a more traditional dedicated broker network model, one benefit of which is the lack of additional network and computation overhead associated with searching a distributed index to perform publish/subscribe matching. The model in PADRES can achieve lower delivery latencies when there are no failures, but still fall back on alternate path publication routing in case of congestion or failure. Admittedly, the DHT protocols, designed for more hostile network, tend to be more fault-tolerant than the algorithms in this paper which assume a more reliable, dedicated broker network.


Publish/subscribe systems have been developed for even more adverse environments such as mobile ad-hoc networks (MANET). These networks are inherently cyclic but the protocols (Lee, 2000; Petrovic, 2005) require periodic refreshing of state among brokers due to the unreliability of nodes and links, an overhead that is not required by the work in this paper. As well, MANET brokers can exploit wireless broadcast channels to optimize message forwarding. For example, brokers in ODMRP (Lee, 2000) do not maintain forwarding tables, but only record if they lie on the path between sources and sinks in a given group. Brokers simply broadcast messages to their neighbors (discarding duplicates) until the message reaches the destinations. The protocols in PADRES, on the other hand, cannot rely on broadcast transmission and also explicitly attempt to avoid duplicate message delivery. As well, ODMRP does not support the more complex content-based semantics.

**Composite Subscriptions:** A *composite subscription* correlates publications over time, and describes a complex event pattern. Supporting an expressive subscription language and determining the location of composite event detection in a distributed environment are difficult problems. CEA (Pietzuch, 2004) proposes a Core Composite Event Language to express concurrent event patterns. The CEA language is compiled into automata for distributed event detection supporting regular expression-type patterns. CEA employs polices to ensure that mobile event detec-

tors are located at favorable locations, such as close to event sources. However, CEA's distribution policies do not consider the alternate paths and the dynamic load characteristics of the overlay network.

One of the key challenges in supporting composite subscriptions in a distributed publish/subscribe system is determining how the subscription should be decomposed and where in the network event collection and correlation should occur. While this problem is similar to query plan optimization in distributed DBMS (Özsu, 1999) and distributed stream processing (Kumar, 2006), data in a relation or a stream have a known schema which simplifies matching and routing. Moreover, a database query is evaluated once against existing data, while a subscription is evaluated against publications over time. This may result in different optimization strategies and cost models. In the IFLOW (Kumar, 2006) distributed stream processing engine, a set of operators are installed in the network to process streams. IFLOW nodes are organized in a cluster hierarchy, with nodes higher in the hierarchy assigned more responsibility, whereas in PADRES (Li, 2005), brokers have equal responsibility.

**Load Balancing:** Although distributed content-based publish/subscribe systems have been widely studied, load balancing was never directly addressed. The following are various related works that propose load balancing techniques in other publish/subscribe approaches.

Meghdoot (Gupta, 2004) distributes load by replicating or splitting the locally heaviest loaded peer in half to share the responsibility of subscription management or event propagation. Such partitioning and replication schemes are common load balancing techniques used in other DHT-based publish/subscribe systems (Aekaterinidis, 2006; Zhu, 2007). In general, their load sharing algorithm is only invoked upon new peers joining the system and peers are assumed to be homogeneous. (Chen, 2005) proposed a dynamic overlay reconstruction algorithm called *Opportunistic Overlay* that reduces end-to-end delivery delay and also performs load distribution on the CP  lization as a secondary requirement. Load balancing is triggered only when a client finds another broker that is closer than its home broker. It is possible that subscriber migrations may overload a non-overloaded broker if the load requirements of the migrated subscription exceed the load-accepting broker's processing capacity. Subscription clustering is another technique to achieve load balancing in content-based publish/subscribe systems (Wong, 2000; Riabov, 2002; Riabov, 2003; Casalicchio, 2007). Subscriptions of similar interests are clustered together at different servers to distribute load. However, architecturally, this technique is not applicable to filter-based but only to multicast-based publish/subscribe systems. PADRES differs from the prior three solutions by proposing a distributed load balancing algorithm for non-DHT filter-based publish/subscribe systems that accounts for heterogeneous brokers and subscribers, and distributes load evenly onto all resources in the system without requiring new client joins. As well, a subscriber migration protocol enforces end-user transparency and best-effort delivery to minimize message loss.

**Fault-tolerance:** Most of the previous work in the fault-tolerant publish/subscribe literature take a best-effort approach and fail to provide strict publication delivery guarantees. Gryphon (Bhola, 2002) is one of the few systems that ensure a similar level of reliability as in PADRES. However, in order to achieve  $\delta$ -fault-tolerance, the routing information of each Gryphon broker must be replicated on  $\delta+1$  other nodes. This design is prone to over provisioning of resources. On the other hand, if a load balancing mechanism is present to improve the node utilization, there are chances that failure of some nodes overwhelms their non-faulty peers (replicas). PADRES improves on these shortcomings by not requiring the assignment of additional replicas. Moreover, the incoming traffic to non-faulty nodes in our system is independent of the number of failures. This implies that in presence of failures, non-faulty peers observe a much lower load increase which is the result of an increase on the number of outgoing messages only. Snoeren et al. (Snoeren, 2001) propose another approach to implement a fault-tolerant P/S system which is based on the construction of several *disjoint* paths between each pair of publishers and subscribers. Publications messages are concurrently forwarded on all disjoint paths enabling the system to tolerate



multiple failures. However, this implies that even in non-faulty conditions the publication traffic can be several times the traffic of the system without fault-tolerance support. In many cases, this overhead makes this scheme impractical. On the other hand, this approach has the advantage of minimizing the impact of failures on publication delivery delay, as the delay is equal to the delivery delay of publications propagated on the fastest path.

## 1.9 Conclusions

This chapter gave an overview of the PADRES content-based publish/subscribe system. It described the message format, subscription language, and data model used in the system. Content-based routing was discussed with particular emphasis on how routing is enabled in cyclic overlays. Cyclic overlays provide redundancy in routes between sources and sinks and thus produce alternative paths between them. Therefore, unlike acyclic overlays, cyclic overlays can be more easily exploited to design a system that can tolerate load imbalances, congestion, and broker failures.

In addition to the ability to route around the affected parts of the network, PADRES also implements other efficient load balancing and recovery algorithms to handle load imbalances and broker failures. These techniques were described in details in this chapter.

To exemplify how content-based publish/subscribe can be used in practice, we presented a detailed discussion of example applications that benefit from the content-based nature of the paradigm. These applications can also take advantage of the scalability and robustness of PADRES.

The PADRES code base is released under an open source license (<http://padres.msrg.utoronto.ca>). The release comprises the PADRES publish/subscribe broker, a client library that allows third party applications to make use of PADRES, a monitoring client, a set of application demonstrations, and the PANDA deployment tool (<http://research.msrg.utoronto.ca/Padres/PadresDownload>). A user and developer guide is also available.

The PADRES publish/subscribe broker is based on a content-based matching engine that supports the subscription language described in Section 1.2.1, including atomic subscriptions, the various forms of historic subscriptions, composite subscriptions with conjunctive and disjunctive operators, the *isPresent* operator, variable bindings, and event correlation with different consumption policies. The PADRES broker was based on the Jess rule engine (Friedman-Hill, 2003), not distributed with our release. The released broker is still compatible with the Jess rule engine, which can be used instead of the matching engine distributed in the release. Most of the results reported in our publications are based on the Jess rule engine as the content-based matching and event correlation mechanism for PADRES. All features described in this chapter, except the load balancing and the fault tolerance features, are included in the PADRES open source release.

PADRES is used in several research and development projects. In the eQoSSystem project with IBM (Jacobsen, 2006; Muthusamy, 2007; Chau, 2008), PADRES constitutes the enterprise service bus that enables the monitoring and enforcement of SLAs of composite applications and business processes in service oriented architectures. In collaborations with Bell Canada (Jacobsen, 2007), PADRES serves to study enterprise application integration problems pertaining to the integration and execution of business processes across existing integration hubs. In collaborations with CA and Sun Microsystems, PADRES is used to explore the event-based management of business processes and business activity monitoring (Li, 2007). In collaborations with the Chinese Academy of Sciences, PADRES is used for service selection (Hu, 2008) and for resource and service discovery in computational Grids (Yan, 2009).



## ACKNOWLEDGEMENT

The PADRES research project was sponsored by CA, Inc., Sun Microsystems, the Ontario Centers of Excellence, the Canada Foundation for Innovation, the Ontario Innovation Trust, the Ontario Early Researcher Award, and the Natural Sciences and Engineering Research Council of Canada.

The completion of the research described in this chapter was also made possible in part thanks to the support through Bell Canada's Bell University Laboratories R&D program, the IBM's Center for Advanced Studies and various IBM Faculty Awards.

The authors would like to thank Serge Mankovskii, CA, Inc. for valuable input to the research presented in this chapter.

The authors would also like to acknowledge the contributions of other past and present members of the PADRES research team for their contributions to the project. This includes Chen Chen, Amer Farroukh, Eli Fidler, Gerald Chen, Ferdous Jewel, Patrick Lee, Jian Li, David Matheson, Pengcheng Wan, Alex Wun, Shuang Hou, Songlin Hu, Naweed Tajuddin, and Young Yoon.

## REFERENCES

Adler, M., Ge, Z., Kurose, J., Towsley, D., & Zabele, S. (2001). Channelization Problem in Large Scale Data Dissemination. *Proceedings of the Ninth International Conference on Network Protocols*(pp.100-110). IEEE Computer Society. Washington, DC, USA.

Aekaterinidis, I., & Triantafillou, P. (2006). PastryStrings: A comprehensive content0based publish/subscribe DHT network.

Andrews, T. (2003). Business Process Execution Language for Web Services. Retrieved Oct. 31 2006 from <http://www.ibm.com/developerworks/library/specification/ws-bpel/>

Ashayer, G., Leung, H., & Jacobsen, H.- A. (2002). Predicate Matching and Subscription Matching in publish/subscribe Systems. *Proceedings of the 22nd International Conference on Distributed Computing Systems*(pp. 539 - 548). IEEE Computer Society. Washington, DC, USA.

Bhola, S., Strom, R. E., Bagchi, S., Zhao, Y., & Auerbach, J.S. (2002). Exactly-once Delivery in a Content-based Publish-Subscribe System. *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, (pp 7-16). IEEE Computer Society. Washington, DC, USA

Birman, K. P., Hayden, M., Ozkasap, O., Xiao, Z., Budiu, M., & Minsky, Y. (1999). Bimodal multicast. *ACM Transaction on Computer Systems*, 17(2), 41-88.

Bittner, S. & Hinze, A. (2007). The arbitrary Boolean publish/subscribe model: making the case. *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, (pp 226 - 237). ACM, New York, NY, USA

Carzaniga, A., Rosenblum, D. S., & Wolf, A. L. (2001). Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3), 332-383

Carzaniga, A., & Wolf, A. L. (2003). Forwarding in a Content-Based Network. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications* (pp 163-174). ACM, New York, NY, USA

- Casalicchio, E., & Morabito, F. (2007). Distributed subscriptions clustering with limited knowledge sharing for content-based publish/subscribe systems. *Proceedings of Network Computing and Applications*, (pp 105-112). Cambridge, MA, USA
- Castro, M., Druschel, P., Kermarrec, A. M., & Rowstron, A. (2002). SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8), 100-110.
- Chau, T., Muthusamy, V., Jacobsen, H.A.; Litani, E., Chan, A., & Coulthard, P. (2008). Automating SLA modeling. *Proceedings of the 2008 conference of the Centre for Advanced Studies on Collaborative Research*, Richmond Hill, Ontario, Canada.
- Chen, Y. & Schwan, K. (2005). Opportunistic Overlays: Efficient Content Delivery in Mobile Ad Hoc Networks. *Proceedings of the 6th ACM/IFIP/USENIX International Middleware Conference* (pp 354-374), Springer.
- Cheung, A., & Jacobsen, H.- A. (2006). Dynamic Load Balancing in Distributed Content-based Publish/Subscribe. *Proceedings of the 7th ACM/IFIP/USENIX International Middleware Conference*(pp 249-269) , Springer.
- Cheung, A., & Jacobsen, H.- A. (2008). Efficient Load Distribution in Publish/Subscribe. *Technical report*, Middleware Systems Research Group, University of Toronto.
- Cugola, G., Nitto, E.D., & Fuggetta, A. (2001). The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9), 827 – 850, IEEE Press Piscataway, NJ, USA
- Deering, S., & Cheriton, D. R. (1990). Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2), 85-111.
- Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A. M. (2003). The many faces of publish/subscribe. *ACM Computing Survey*, 35(2), 114-131
- Fabret, F., Jacobsen, H.- A., Llibat, F., Pereira, J., Ross, K. A., & Shasha, D. (2001). Filtering algorithms and implementation for very fast publish/subscribe systems. *Proceedings of the 2001 ACM SIGMOD international conference on management of data* (pp 115-126), ACM New York, NY, USA
- Fawcett, T. & Provost, F., Activity monitoring: Noticing interesting changes in behavior. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp 53-62), ACM New York, NY, USA
- Fiege, L., Mezini, M., Mühl, G. & Buchmann, A. P., (2002). Engineering Event-Based Systems with Scopes. *Proceedings of the 16th European Conference on Object-Oriented Programming* (pp 309-333), Springer.
- Forgy, C. L., (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19(1),17-37

- Friedman-Hill, E. J., (2003). Jess, The Rule Engine for the Java Platform.  
<http://herzberg.ca.sandia.gov/jess/>
- Gupta, A., Sahin, O. D., Agrawal, D., & Abbadi, A. E., (2004). Meghdoot: Content-Based publish/subscribe over P2P Networks. *Proceedings of the 5th ACM/IFIP/USENIX International Middleware Conference* (pp 254-273), Springer.
- Hu, S., Muthusamy, V., Li, G. & Jacobsen, H.- A., (2008). Distributed Automatic Service Composition in Large-Scale Systems. *Proceedings of the second international conference on Distributed event-based systems* (pp 233-244), ACM New York, NY, USA
- IBM. (2003). Web Service Level Agreements (WSLA) Project. Retrieved July 12th 2007 from <http://www.research.ibm.com/wsla/>.
- Jacobsen, H.- A., (2004). PADRES User Guide. Retrieved July 19th 2006 from <http://research.msrg.utoronto.ca/Padres/UserGuide>.
- Jacobsen, H.- A., (2006). eQoSystem. <http://research.msrg.utoronto.ca/Eqosystem>
- Jacobsen, H.- A., (2007). Enterprise Application Integration.  
<http://research.msrg.utoronto.ca/EAI/>
- Koenig, I. (2007). Event Processing as a Core Capability of Your Content Distribution Fabric. *Gartner Event Processing Summit*, Orlando, Florida.
- Kumar, V., & Cai, Z. (2006). Implementing Diverse Messaging Models with Self-Managing Properties using IFLOW. *IEEE International Conference on Autonomic Computing* (pp 243-252). IEEE Computer Society. Washington, DC, USA.
- Lee, S., Su, W., Hsu, J., Gerla, M., & Bagrodia, R., (2000). A performance comparison study of ad hoc wireless multicast protocols. *Proceedings of 9th Annual Joint Conference of the IEEE Computer and Communications Societies* (pp 565-574), IEEE Computer Society. Washington, DC, USA.
- Li, G., Hou, S. & Jacobsen, H.- A. (2005). A Unified Approach to Routing, Covering and Merging in Publish/Subscribe Systems based on Modified Binary Decision Diagrams. *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems* (pp 447-457), Columbus, Ohio, USA
- Li, G., Hou, S. & Jacobsen, H.- A. (2008). Routing of XML and XPath Queries in Data Dissemination Networks. *Proceedings of the 28th IEEE International Conference on Distributed Computing Systems* (pp 627-638), Beijing, China.
- Li, G. & Jacobsen, H.- A. (2005). Composite Subscriptions in Content-Based publish/subscribe Systems. *Proceedings of the 6th ACM/IFIP/USENIX International Middleware Conference* (pp 249-269), Springer.
- Li, G., Cheung, A., Hou, S., Hu, S., Muthusamy, V., Sherafat, R., Wun, A., Jacobsen, H.- A. & Manovski, S. (2007). Historic data access in publish/subscribe. *Proceedings of the 2007 inaugural international conference on Distributed event-based systems* (pp 80-84), Toronto, Ontario, Canada.

- Li, G., Muthusamy, V., & Jacobsen, H.- A. (2007). NIÑOS: A Distributed Service Oriented Architecture for Business Process Execution. *Technical report*, Middleware Systems Research Group, University of Toronto.
- Li, G., Muthusamy, V., & Jacobsen, H.- A. (2008). Adaptive content-based routing in general overlay topologies. *Proceedings of the 9th ACM/IFIP/USENIX International Middleware Conference* (pp 249-269), Springer.
- Li, G., Muthusamy, V., & Jacobsen, H.- A. (2008). Subscribing to the past in content-based publish/subscribe. *Technical report*, Middleware Systems Research Group, University of Toronto.
- Liu, H., & Jacobsen, H.A. (2002). A-ToPSS: A Publish/Subscribe System Supporting Approximate Matching. *Proceedings of 28th International Conference on Very Large Data Bases* (pp 1107-1110), Hong Kong, China.
- Liu, H., & Jacobsen, H.A. (2004). Modeling uncertainties in publish/subscribe systems. *Proceedings of the 20th International conference on Data Engineering* (pp 510-522), Boston, MA, USA.
- Liu, H., Muthusamy, V., & Jacobsen, H.A. (2009). Predictive Publish/Subscribe Matching. *Technical report*, Middleware Systems Research Group, University of Toronto.
- Liu, H., Ramasubramanian, V. & Sirer, E. G., (2005). Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement* (pp 3-3), USENIX Association Berkeley, CA, USA.
- Mühl, G., (2002). *Large-scale content-based publish/subscribe systems*. Unpublished doctoral dissertation, Darmstadt University of Technology, Germany.
- Mukherjee, B., Heberlein, L. T., & Levitt, K. N. (1994). Network intrusion detection. *IEEE Network*, 8(3), 26-41
- Muthusamy, V., & Jacobsen, H.- A. (2005). Small-scale Peer-to-peer Publish/Subscribe. *Proceedings of the MobiQuitous Conference* (pp 109-119), ACM New York, NY, USA
- Muthusamy, V., & Jacobsen, H.- A. (2007). Infrastructure-less Content-Based Pub. *Technical report*, Middleware Systems Research Group, University of Toronto.
- Muthusamy, V., Jacobsen, H.- A., Coulthard, P., Chan, A., Waterhouse, J. & Litani, E. (2007). SLA-Driven Business Process Management in SOA. *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research* (pp 264-267), Richmond Hill, Ontario, Canada.
- Muthusamy, V., & Jacobsen, H.- A. (2008). SLA-driven distributed application development. *Proceedings of the 3rd Workshop on Middleware for Service Oriented Computing* (pp 31-36), Leuven, Belgium.

- Nayate, A., Dahlin, M. & Iyengar, A., (2004). Transparent information dissemination. *Proceedings of the 5th ACM/IFIP/USENIX International Middleware Conference* (pp 212 - 231), Springer.
- Oki, B., Pfluegl, M., Siegel, A. & Skeen, D. (1993). The Information Bus: an architecture for extensible distributed systems. *Proceedings of the fourteenth ACM symposium on Operating systems principles* (pp 58-68). New York, NY, USA.
- Opyrchal, L., Astley, M., Auerbach, J., Banavar, G., Strom, R., & Sturman D., (2000). Exploiting IP multicast in content-based publish-subscribe systems. *IFIP/ACM International Conference on Distributed systems platforms*, (pp 185-207), Springer-Verlag New York, Inc.
- Ostrowski, K., & Birman, K., (2006). Extensible Web Services Architecture for Notification in Large-Scale Systems. *Proceedings of the IEEE International Conference on Web Services* (pp 383-392), IEEE Computer Society Washington, DC, USA.
- Özsu, M. T., & Valduriez, P., (1999). *Principles of Distributed Database Systems*. Prentice Hall
- Petrovic, M., Muthusamy, V., & Jacobsen, H.- A. (2005). Content-based routing in mobile ad hoc networks. *Proceedings of the Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services* (pp 45-55), San Diego, CA, USA.
- Pietzuch, P. R., & Bacon, J. (2002). Hermes: A Distributed Event-Based Middleware Architecture. *Proceedings of the 22nd International Conference on Distributed Computing Systems*, (pp 611-618), IEEE Computer Society.
- Pietzuch, P. R., Shand, B., & Bacon, J. (2004). Composite Event Detection as a Generic Middleware Extension. *IEEE Network Magazine, Special Issue on Middleware Technologies for Future Communication Networks* (pp 44-55), IEEE Computer Society.
- PlanetLab. (2006). <http://www.planet-lab.org/>.
- Powell, D, (1996). Group communication. *Communications of the ACM*, 39(4), 50-53, ACM New York, NY, USA.
- Riabov, A., Liu, Z., Wolf, J. L., Yu, P. S., & Zhang, L. (2002). Clustering algorithms for content-based publication-subscription systems. *Proceedings of the 22nd International Conference on Distributed Computing Systems* (pp 133-142), IEEE Computer Society.
- Riabov, A., Liu, Z., Wolf, J. L., Yu, P. S., & Zhang, L. (2003). New Algorithms for Content-Based Publication-Subscription Systems. *Proceedings of the 23rd International Conference on Distributed Computing Systems* (pp 678- 686), IEEE Computer Society.
- Rose, I., Murty, R., Pietzuch, P., Ledlie, J., Roussopoulos, M., & Welsh, M., (2007). Cobra: Content-based Filtering and Aggregation of Blogs and RSS Feeds. *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation* (pp 231-245 ), Cambridge, MA.

- Schuler, C., Schuldt, H., & Schek, H. J., (2001). Supporting Reliable Transactional Business Processes by publish/subscribe Techniques. *Proceedings of the Second International Workshop on Technologies for E-Services* (pp 118-131), Springer-Verlag London, UK.
- Sherafat Kazemzadeh, R. & Jacobsen, H.-A., (2007).  $\delta$ -Fault-Tolerant Publish/Subscribe systems. *Technical report*, Middleware Systems Research Group, University of Toronto.
- Sherafat Kazemzadeh, R. & Jacobsen, H.-A., (2008). Highly Available Distributed Publish/Subscribe Systems. *Technical report*, Middleware Systems Research Group, University of Toronto.
- Snoeren, A. C., Conley, K., & Gifford, D. K., (2001). Mesh-based content routing using XML. *ACM SIGOPS Operating Systems Review*. 35(5), 160-173, ACM New York, NY, USA.
- Tock, Y., Naaman, N., Harpaz, A., & Gershinsky, G., (2005). Hierarchical Clustering of Message Flows in a Multicast Data Dissemination System. *Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems* (pp 320-326), ACTA Press.
- Triantafillou, P., & Economides, A., Subscription Summarization: A New Paradigm for Efficient publish/subscribe Systems. *Proceedings of the 24th International Conference on Distributed Computing Systems*, (pp 562-571), IEEE Computer Society.
- Wong, T., Katz, R. H., & McCanne, S., (2000). An evaluation of preference clustering in large-scale multicast applications. *Proceedings of the conference on computer communications* (pp 451-460), IEEE Computer Society.
- Yan, W., Hu, S., Muthusamy, V., Jacobsen, H.-A. & Zha, L, (2009). Efficient event-based resource discovery. *Proceedings of the 2009 inaugural international conference on Distributed event-based systems*, Nashville, Tennessee, USA.
- Zhu, Y., & Hu, Y., (2007). Ferry: A P2P-Based Architecture for Content-Based publish/subscribe Services. *IEEE Transaction on Parallel Distributed Systems*, 18(5), 672-685, IEEE Computer Society.