Incorporating an XML Matching Engine in Distributed Brokering Systems

Shrideep Pallickara, Geoffrey Fox and Marlon Pierce
Community Grid Labs, Indiana University
{spallick,gcf,mpierce}@indiana.edu

1.0 Introduction

As systems such as peer-to-peer, Web Services, and Grid systems proliferate the Internet there have also been a significant increase in the number of devices that are networked to the Internet. In a trend that will continue in the coming years the number, type and complexity of the interactions that entities seek with each other will continue to increase. This is also coupled with an increase in the volume of information that would be generated by entities. XML (short for eXtensible Markup Language) [1] is a structured document format, in that it represents not only the information to be exchanged, but the metadata encapsulating its meaning, and the structure of the information to be exchanged. XML's data encapsulation properties allow us to access relevant fields in the performance data easily. Tags, attributes and element structures provide context information, which can then be used to interpret the meaning of the content which provides intelligent data mining. The XML markup language has been used increasingly for data interchange between applications and entities. More recently XML based standards have been proposed by the W3C to support the Web Services concept where XML has been used to describe services, and carry invocations to these services. SOAP [2] provides a general purpose message envelop that may be specialized to the problem of transmitting remote procedure calls and responses

The process of matching refers to the process of computing destinations from messages or events (essentially time-stamped messages) encapsulating interactions and communications. The matching process usually operates on a set of constraints that are either statically specified by entities or dynamically generated by services. In this paper we discuss issues pertinent to the problem of matching XML events to specified constraints.

The XML matching problem for distributed brokering systems is an important one for three distinct reasons. First, as the scale of the system (quantified in terms of number of entities, devices and systems) increases, any infrastructure providing the system backbone will gravitate towards a distributed solution. Considerations pertaining to single-point-of failures, scaling, reliability and availability are the overriding factors, which dictate this choice.

Second, as entities exchange information and interact with each other in XML encapsulated communications, they would also incorporate increasingly complex constraints on the interactions that they are interested in. These constraints need to be based on emerging standards to retrieve information from XML documents.

Finally, and more subtly, such an XML matching engine would provide the underpinnings to route Web Service invocations. The system, which currently would be used to route the right XML encapsulated content to the interested entities, would also then be used to search, discover and subsequently route service interface descriptions and prepared service invocations over the distributed brokering system. Thus the distributed matching system forms the substrate on which we may build both lightweight distributed information systems and location independent services.

We base our investigations in the context of our advanced research prototype NaradaBrokering [3-8]. In brief summary, NaradaBrokering is a distributed system for managing messages and events (messages with timestamps). NaradaBrokering can be used to process and route arbitrary messages, but here we consider the special case of XML events/messages. Besides providing a strategy for incorporating a distributed XML matching within a distributed messaging infrastructure, this paper investigates the following issues.

1. *Study different styles of matching*: There are two different styles of matching, which though based upon the same matching engine are substantially different. First, there is the matching of a given event to a set of maintained constraints resulting in the event being routed to valid destinations. Then, there is the matching of query to a set of stored XML events.
2. *Feasibility of the matching operations*: In each of the cases, which we alluded to earlier, the costs associated with the operations may be compared. This would allow us to determine whether premiums need to be associated with different matching strategies.

3. *Recommendations*: Based on our research, we also incorporate a set of recommendations and caveats, if any, that must be taken into account while building systems that build on the work outlined in this paper.

2.0 Related Work

A body of work exists in publish/subscribe messaging that operates on computing destinations from specialized events. The strategy adopted in Elvin [9] to compute destinations relies on converting subscriptions into a deterministic finite state automaton, which can sometimes lead to explosive search spaces. In Sienna [10] optimization strategies include assembling patterns of notifications as close as possible to the publishers, while multicasting notifications as close as possible to the subscribers. Gryphon [11] provides support for content-based generalized matching algorithm that operates on <tag,value> specified constraints.

As far as the processing and routing of XML encapsulated messages is concerned, peer-to-peer (P2P) systems [12] have deployed such services. The JXTA [13] (from juxtaposition) project at Sun Microsystems is a research effort to support large-scale P2P infrastructures. P2P interactions, described in XML, are propagated by a simple forwarding by peers and specialized routers known as rendezvous peers. These interactions are attenuated by having TTL (time-to-live) indicators and also by the peer traces that eliminate the continuous echoing problem caused by traces in peer connectivity. [14] suggests an approach to using the Java Message Service [15] for transporting XML content as a strategy to route B2B data interchange. The strategy suggested here is to encapsulate content in the JMS `TextMessage` primitive. The slash-separated string topic contained in the message is what is then used to route content.

3.0 NaradaBrokering: Organization and Overview

In this section we present an overview of the NaradaBrokering system. Specifically we provide a brief review of the broker organization scheme and the routing strategy within the broker network. To address the issues [16] of scaling, load balancing and failure resiliency, NaradaBrokering is implemented on a network of cooperating brokers. In NaradaBrokering we impose a hierarchical structure on the broker network, where a broker is part of a cluster that is part of a super-cluster, which in turn is part of a super-super-cluster and so on. Clusters comprise strongly connected brokers with multiple links to brokers in other clusters, ensuring alternate communication routes during failures. This organization scheme results in "small world networks" [17,18] where the average communication pathlengths between brokers increase logarithmically with geometric increases in network size, as opposed to exponential increases in uncontrolled settings. This distributed cluster architecture allows NaradaBrokering to support large heterogeneous client configurations that scale to arbitrary size. Within every unit, there is at least one unit-controller, which provides a gateway to nodes in other units. For example in figure 1 cluster controller node **20** provides a gateway to nodes in cluster **m**.

Creation of broker network maps (BNMs) and the detection of network partitions are easily achieved in this topology. We augment the BNM hosted at individual brokers to reflect the cost associated with traversal over connections, for e.g. intra-cluster communications are faster than inter-cluster communications. The BNM can now be used not only to compute valid paths but also for computing shortest paths. Changes to the network fabric are propagated only to those brokers that have their broker network view altered.

Events routed within NaradaBrokering have an implicit or explicit destination list, comprising clients, associated with it. The brokering system as a whole is responsible for computing broker destinations (targets) and ensuring efficient delivery to these targeted brokers en route to the intended client(s). Events as they pass through the broker network are be updated to snapshot its dissemination within the network. The event dissemination traces eliminate continuous echoing and in tandem with the BNM – used for computing shortest paths – at each broker, is used to deploy a near optimal routing solution. The routing [19] is near optimal since for every event, the associated targeted set of brokers comprises only those that are/should-be involved in the disseminations. Furthermore, every broker, either targeted or en route to one, computes the shortest path to reach target destinations while employing only those links and brokers that have not failed or been failure-suspected.

4.0 XPath: The strategy to query

XPath [20] is a query language that searches for, locates, and identifies parts of an XML document. In this case there is no hint such as "topic" contained in the XML event and the query needs to be matched with the entire XML event. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath

operates on the abstract, logical structure of an XML document, rather than its surface syntax. XPath use a path notation, as in URLs, for navigating through the hierarchical structure of an XML document. [21] provides query strategies in the context of a P2P Grid. Though our current investigations are based on XPath, future work will address related specifications such as XQuery, XQL and XML-SQL.

5.0 The Distributed XML Matching Engine

The most fundamental operation in the XML matching process is the evaluation of whether the XML document satisfies the constraint specified in the XPath query. When the constraint is satisfied we say that the document matches the query. Depending on the type of interaction, which the entity seeks with the system, the matching process is subtly different.

The interactions which entities seek with system fall into two broad categories. First, entities could specify constraints that real time events (encapsulating XML content) need to satisfy before being routed to them, for e.g. when bids on an auctioned entity goes below a certain value, one might want to be notified. We refer to the constraints, which entities specify as subscriptions. These constraints are expressed as XPath queries. Effectively, entities subscribe to any incoming XML events that match their XPath constraints. Entities propagate this information, along with their destination information, in profiles.

The second interaction mode is based on an entity being interested in locating a resource or in retrieving information from a stored set of events. Entities can expose information pertaining to the resources they share. This information, an advertisement, is an XML based description of the resource/service that can be utilized by entities also includes the advertising entity's destination information. Entities may also incorporate authorization related information in these advertisements. In the case of advertisements, they identify where the resource might be accessed from. While retrieving a set of events, usually recovery events after a prolonged disconnect, entities propagate queries (along with the entity's destination) to relevant locations within the broker network. Matched XML events would then be routed to the destination contained in the query.

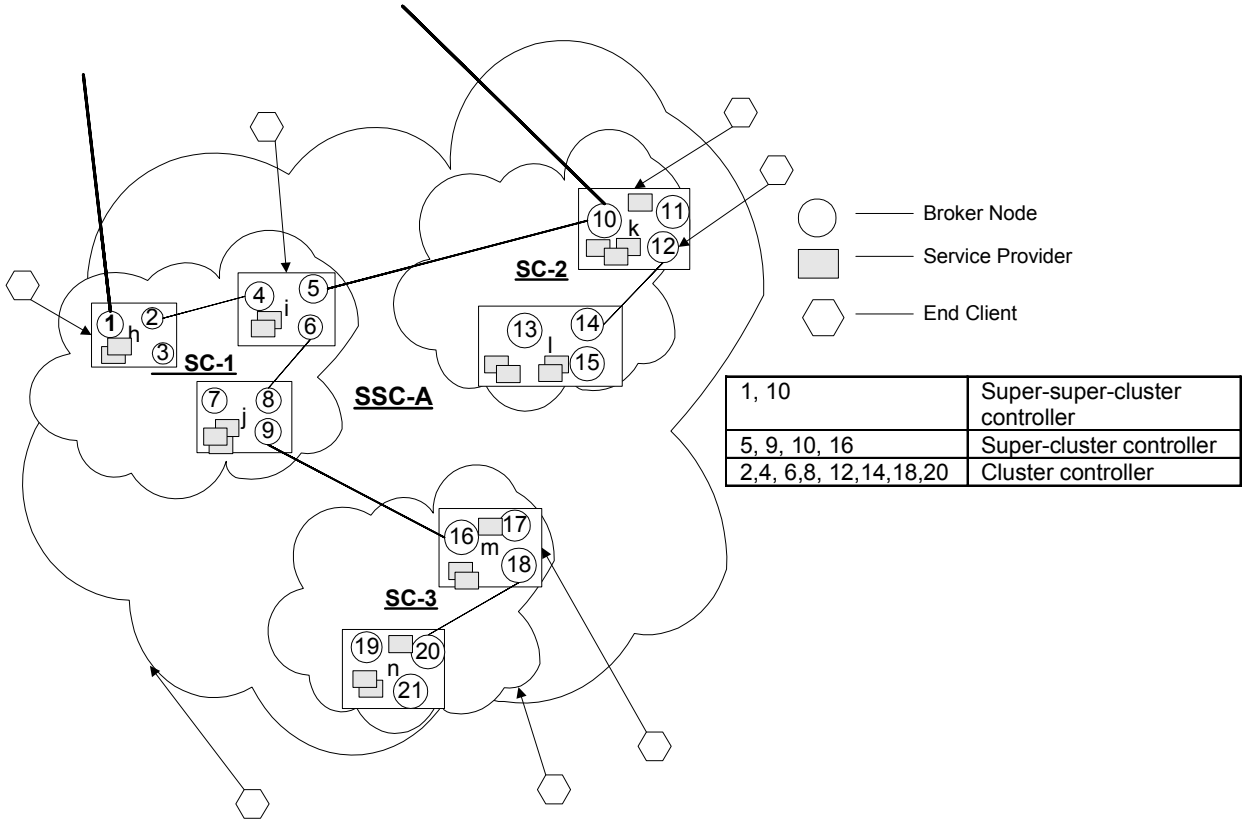

Figure 1: A scheme for matching XML events

In the first case the system serves as an entity matching real-time XML events. Efficient means of doing this are discussed in Section 5.2. In the second case, the system is being used as a light-weight distributed XML database by matching against stored XML events for recovering clients or matching XML advertisements. This is discussed in detail in Section 5.3.

The XML matching engine is responsible for optimizing the efficiency associated with the matching operations in each of the cases discussed earlier. Since the computation of destinations and identification of the right resources is in itself a distributed process, the matching engine at each node needs to exploit the topology, the propagation of profiles/advertisements and the routing algorithms in place. The matching engine needs to ensure that it computes destinations so that all entities interested in the event receive the event. Similarly, if a valid resource exists for an entity's query, the associated advertisement needs to be routed to the querying entity.

5.1 Organizing Profiles and Advertisements

In NaradaBrokering the hierarchical organization of broker nodes lends itself to very efficient organization of profiles/advertisements. Figure 1 depicts a sub-system comprising of a super-super-cluster **SSC-A** with 3 super-clusters **SC-1**, **SC-2** and **SC-3** each of which have clusters that in turn are comprised of broker nodes. A unit controller keeps tracks of all the profiles/advertisements propagated by, and within, its sub-units. Thus, a super-cluster controller keeps track of all the profiles/advertisements propagated by cluster controllers within the super cluster that it controls. Similarly, a cluster controller keeps track of all the profiles/advertisements propagated by the broker nodes within the cluster that it controls.

Thus, in figure 1, super-super-cluster controller nodes **1** and **10** keep track of all profiles/advertisements propagated by all the nodes (**1** through **21)** in super-super-cluster **SSC-A**, while cluster controller node **19** of cluster **n** would keep track of profiles/advertisements propagated by nodes **19**,**20**, **21** in cluster **n**. Since a unit controller operates and communicates only with sub-unit destinations, all profiles/advertisements are stored at the controllers as if they originated at specific sub-units. Thus, for an advertisement propagated by a service connected to node **21**, the advertisement is stored at the cluster controller node **20** to reflect that it came from node **21**, while the super-cluster controller node **16** registers it as having coming from cluster **n**, with the super-super-cluster controller nodes **1**, **10** registering it as having originated in super-cluster **SC-3**.

5.2 Matching Profiles

This section addresses the problem of real time matching operations needed to route XML events to their final destinations. Here XML events propagate through the system and must be matched immediately. Here we substitute the entire XML message for the publish-subscribe notion of topics. Any final destination in the system will receive any published XML events that match their XPath-expressed subscriptions. The XML Matching Engine will identify all destinations that have matching XPath constraints. After the final destinations for a particular event are determined, the system must compute the efficient routes to these destinations. The routing strategy is discussed briefly in Section 3.0.

For several reasons we limit the number of sub-units within a unit to 32. For example, in Figure 1, a cluster controller such node 20 in Figure 1 can manage up to 32 nodes in the cluster. By assigning each sub-unit a unique position in a 32-bit vector, in a system comprising of super-super-clusters, any node (out of a possible 32x32x32x32=1,048,576 nodes) can be uniquely represented by 128-bits (4 integers). For example, in Figure 1, node 19 may be associated with the integer 00…001..00 while node 20 might be associated with 00…010..00. If both nodes should receive an event, then the destination list is the sum (bitwise OR) of these two nodes 00…011…00. This provides a rather compact representation for distribution traces and computed destinations associated with various interactions.

In our strategy for hierarchical calculation of destinations associated with XML events, a unit controller computes sub-unit destinations based on profiles stored at its nodes. These sub-unit destinations are represented in a 32-bit integer. Thus, we have a super-cluster controller, computing cluster destinations and a cluster controller computing broker destinations.

Once a profile is successfully matched to an event, the destination associated with the profile is added to the computed destination. When other profiles are being matched against the event, a check is made to see if the destination associated with the profile is already in the list of computed destinations (a bit-wise AND operation

yields a non-zero value if it is). If it is, the matching process is suspended for this profile, since it would yield a destination that already exists in the computed destinations. If the destination contained in the profile is a different one, the profile is matched with the event. If there is a match the associated destination is added (a bitwise OR operation) to the computed destination list. This scheme substantially reduces the number of matching operations that need to be performed.

A similar strategy is employed by brokers matching events to attached clients (see Figure 1). Of course in this case there is no limit on the number of clients that can be attached to a broker and the number of matching operations that need to be performed is not reduced, as substantially, as in the controller cases.

5.3 Matching Advertisements

This section describes the process by which we match queries to stored XML documents and advertisements. This is equivalent to treating the system as a distributed XML database. In this case the XML advertisements/events could be stored persistently on one or more nodes in the system. When an XPath query is issued by one of the clients connected to a broker, the query propagates hierarchically through the broker network. Thus, the query is propagated by the broker to the cluster controller and from there on to the super-cluster controller and so on. For example, in figure 1, for a query injected into the system from node **21**, the query is propagated to cluster controller node **20**, from there onto super-cluster controller node **16**, and then onto super-super-cluster controller nodes **1**, **10** respectively (though the results returned from **1,10** would be identical, since they have aggregated identical information). When a match is found within the advertisements stored at a controller, the matching advertisements are returned back to the "querying" broker. Since each query also contains information regarding the querying broker, that propagated the XPath query from the client, the system can easily locate and route the matching advertisements to the querying broker. This broker, in turn, keeps track of the client that issued this XPath query and routes the matching advertisements back to the client.

We place one straightforward restriction on query propagation: when a query is passed from a unit to a higher level controller, the controller will match advertisements registered to its subunits *except* the subunit that initiated the query. For example, in figure 1, for a query injected into the system from node **21**, the query is propagated to cluster-controller node **20**. While matching advertisements stored at cluster-controller node **20**, the matching engine will not evaluate the query against advertisements registered to node **21**. This is based the matching process at node **21** would have already computed and route the matching advertisements. Similarly, when the query is propagated to super-cluster controller node **16**, that node ignores advertisements registered to cluster **n**.

In the case of matching a query to a set of stored advertisements, the query is evaluated over the entire set of advertisements (excluding the optimization we discussed earlier). The time associated with evaluating these queries can thus increase substantially with increases in the number of advertisements that need to be matched. The query-based discovery operations generally tend to be far more forgiving than those that demand real-time constraints.

Of course search operations could be modified to return a certain number of matching results. A broker that is not able to satisfy the number specified then propagates the query to the cluster controller by modifying the number of matching results that need to be satisfied. This process repeats itself, propagating to higher order controllers until the number of matches is satisfied or until no more resources can be found.

5.4 Restricting the Scope of Matching

To ensure that resources are not available beyond a certain realm all that needs to be done is to ensure that propagations of advertisements do not go beyond a certain realm or that it reaches only a certain realm. This information could be included in the description of the advertisement/profile. The broker propagating the advertisement could specify that the advertisement/profile should not be propagated beyond the realm of its cluster controller or super-cluster controller.

Similarly ACLs (Access Control Lists) or other identifiers (that would be checked when queries are made) could be included along with the advertisements to ensure that subsequent XPath queries need to satisfy certain constraints before the service is visible to the querying client. Similarly, a depth could be specified in the query for a service and also in the invocations. The advantage of this scheme is that resources located would be closer to the querying entity. This would ensure that the queries do not go beyond the horizons imposed by the search. One could, for example, choose to locate services that are within the range of the cluster/super-cluster controller. Depending on

how far the query propagates, one will have access to services hosted at wider disparate locations. Queries could also be required to possess authentication information along with them or they could be asked to authenticate with the service prior to invoking any operations on the service implementation.

The scheme could also be augmented to cache queries and associated responses. The hierarchical search mechanism that we just outlined could be augmented with P2P search mechanisms between peers at the edge of the network to facilitate a very rich search mechanism.

6.0 Performance Measurements

We now provide some results pertaining to the matching operations that were outlined in preceding sections. These results are computed for stand-alone processes, where we compare the matching times as a function of the number of profiles/advertisements maintained. The results were measured on a machine (1.2 GHz, 256MB RAM, Windows 2000) running the process in a Java-1.4 Sun VM with a high-resolution timer for computing delays. In our implementation we have used Apache's Xalan [22], which implements W3C's XPath Recommendations.

We measure the times required to match an XML event or XPath query to a set of stored profiles or advertisements respectively. The cost of a matching operation is around 3 milliseconds. We also measure the cost associated with matching using the destination optimization strategy discussed in section 5.2. The profiles in the earlier case are evenly distributed over 32 different destinations.

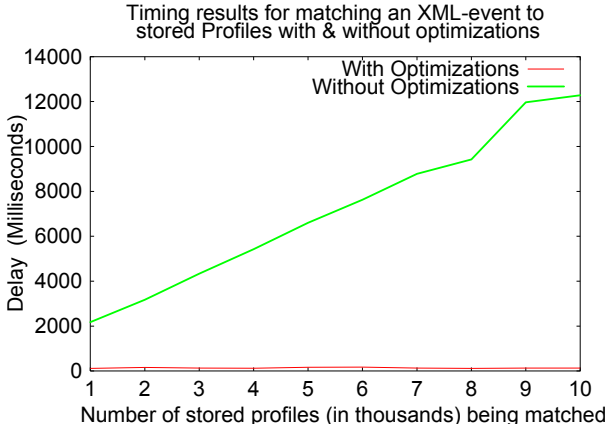

Figure 2: Matching Profiles

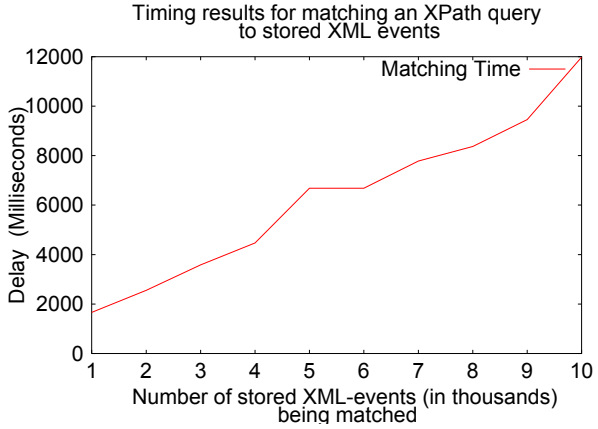

Figure 3: Matching XML-events

Figure 2 contrasts the matching times in the profile matching with/without optimizations for varying number of profiles. With optimizations the matching times varied between 120-170 milliseconds. The results demonstrate that in the scenario outlined earlier, the optimizations improve the performance of matching profiles substantially. In general, in most practical situations it is our conjecture that the performance would be similarly enhanced. Figure 3 depicts the matching times for a query against a set of stored XML events/advertisements. For matching XML advertisements, the performance would vary if it is constrained by the number of matched advertisements/stored-events that need to be included in the query response. Figure 4 outlines the XML type for the stored events, and the type of XPath query used in our experiments.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<menu>
  <softdrinks>
    <brand>Minute Maid</brand>
    <fruit>Apple</fruit>
    <source>Brazil</source>
    <company>Coca Cola</company>
    <price>2.90</price>
    <year>2003</year>
  </softdrinks>
</menu>
```

XPath Query type: **/menu/softdrinks[price>1.80]**

Figure 4: The XML doc and XPath query type

7.0 Conclusions and Future Work

In this paper we presented an approach to matching XML events. As far as we know, NaradaBrokering is the first system to incorporate the notions of distributed XML matching of advertisements and profiles, and the accompanying routing of matched XML content to relevant destinations. We believe that with this integration we

well positioned to exploit efficient, distributed information system for Web Services within the NaradaBrokering Context.

The future work focuses on improving the performance of matching, under two different scenarios. First, two different XPath constraints can be equivalent even if they have been specified in different ways. Equivalence of subscriptions can allow us to relate subscriptions from multiple destinations together. Matching one of these subscriptions, would implying matching of equivalent subscriptions. This would result in the equivalent subscription's destination also being added to the destination list. This obviates the need to evaluate any of the queries with that destination. In certain situations this would lead to substantial gains in response times.

Effective organization of advertisements is the other entry points for reduction of times associated with matching. By related we mean advertisements that have related schema or those whose DOM models have similar nodes and elements. This scheme would also significantly reduce the number of matching operations that need to be performed.

In this paper we presented an approach to matching XML events. As far as we know, NaradaBrokering is the first system to incorporate the notions of distributed XML matching of advertisements and profiles, and the accompanying routing of matched XML content to relevant destinations. We believe that with this integeration we well positioned to exploit the growing of Web Services within the NaradaBrokering Context.

References

1. The Extensible Markup Language, Specifications and Working groups available from http://www.w3.org/XML/
2. Simple Object Access Protocol (SOAP). Available from http://www.w3.org/TR/SOAP/.
3. The NaradaBrokering System http://www.naradabrokering.org
4. A Middleware Framework and Architecture for Peer-to-Peer Grids. Shrideep Pallickara and Geoffrey Fox (To appear) Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.
5. NaradaBrokering: An Event Based Infrastructure for Building Scaleable Durable Peer-to-Peer Grids. Geoffrey Fox and Shrideep Pallickara. Chapter 22 of "Grid Computing: Making the Global Infrastructure a Reality". John Wiley April'03.
6. The Narada Event Brokering System: Overview and Extensions. Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, June 2002. pp 353-359.
7. A Scaleable Event Infrastructure for Peer to Peer Grids. Geoffrey Fox, Shrideep Pallickara and Xi Rao. Proceedings of ACM Java Grande ISCOPE Conference 2002. Seattle, Washington. November 2002.
8. "JMS Compliance in the Narada Event Brokering System." Geoffrey Fox and Shrideep Pallickara. Proceedings of the International Conference on Internet Computing (IC-02). June 2002. pp 391-402.
9. Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content based routing with elvin4. In Proceedings AUUG2K, Canberra, Australia, June 2000.
10. Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Achieving scalability and expressiveness in an internet-scale event notification service. In Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, pages 219–227, Portland OR, USA, 2000.
11. Marcos Aguilera, Rob Strom, Daniel Sturman, Mark Astley, and Tushar Chandra. Matching events in a content-based subscription system. In Proceedings of the 18th ACM Symposium on Principles of Distributed Computing, May 1999.
12. Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. Edited by Andy Oram. O'Rielly Press, CA. March 2001.
13. Sun Microsystems. The JXTA Project and Peer-to-Peer Technology http://www.jxta.org
14. JMS and XML, T.A. Flores On the O'Reilly Network http://www.oreillynet.com/pub/a/onjava/2001/02/15/jms_xml.html.
15. Java Message Service Specification". Mark Happner, Rich Burridge and Rahul Sharma. Sun Microsystems. 2000. http://java.sun.com/products/jms.
16. "An Approach to High Performance Distributed Web Brokering". Fox and Pallickara, ACM Ubiquity 2:38. Nov 2001.
17. "Collective Dynamics of Small-World Networks". D.J. Watts and S.H. Strogatz. *Nature*. 393:440. 1998.
18. "Diameter of the World Wide Web". R. Albert, H. Jeong and A. Barabasi. *Nature* 401:130. 1999.
19. An Event Service to Support Grid Computational Environments Geoffrey Fox and Shrideep Pallickara. Journal of *Concurrency and Computation: Practice & Experience*. Volume 14(13-15) pp 1097-1129.
20. XML Path Language (XPath). Version 1.0. W3C Recommendation. Available from http://www.w3.org/TR/xpath .
21. Peer-to-Peer Grid Databases for Web Service Discovery. Wolfgang Hoschek. To appear in "Grid Computing: Making the Global Infrastructure a Reality". John Wiley April'03.
22. Xalan-Java from Apache Software Foundation. Available from http://xml.apache.org/xalan-j/index.html