Efficient Matching of Events in Distributed Middleware Systems

Shrideep Pallickara and Geoffrey Fox
Community Grids Laboratory
Indiana University, IN. USA 47401
{spallick,gcf}@indiana.edu

Abstract: *Increasingly interactions which entities have with each other are network bound. The entities, with which applications andservices need to interact, span a wide spectrum that includes desktops, PDAs, appliances, and other networked resources. These events encapsulate information pertaining to transactions, data interchange, system conditions and finally the search, discoveryand subsequent sharing of resources. Events have internal or external (system computed) destinations associated with them. In the case of search, discovery and publish/subscribe interactions the system has to efficiently calculate destinations from the corresponding events. This computing of destinations is referred to as matching and is, in itself, a distributed process, which operates on the distributed management of client interests (advertisements and subscriptions). The distributed nature of the underlying messaging infrastructure also mandates an efficient routing engine. Inefficient approaches to either the calculation of, or routing to, destinations can result in unacceptable network degradations.*

In this paper we explore matching, routing and network utilization issues in the context of NaradaBrokering, which provides support for centralized, distributed and P2P interactions. We also discuss the implications, and include results, pertaining to the different matching engines supported within the NaradaBrokering system.

Key words: distributed messaging, publish/subscribe, middleware, matching, event based systems

Reviewed and accepted: 31 Mar. 2004

1. Introduction

The Internet is currently being used to support increasingly complex interactions. The devices, with which applications and services need to interact, span a wide spectrum that includes desktops, PDAs, appliances, and other networked resources. Clients – which abstract users, resources and proxies thereto – within these systems communicate with each other through the exchange of events, which are essentially messages with timestamps. These events encapsulate information pertaining to transactions, data interchange, system conditions and finally the

search, discovery and subsequent sharing of resources. Scaling, availability and fault tolerance requirements entail that the messaging infrastructure hosting these clients, and routing their interactions, be based on a distributed network of cooperating nodes. As the scale of the system increases, effective interactions between clients and services, in these settings, is dictated

not just by the processing power of the nodes hosting a specific service but also by the network cycles expended during these interactions. Events have internal or external (system computed) destinations associated with them. In the case of search, discovery and publish/subscribe interactions, the system has to efficiently calculate destinations from the corresponding events. This computing of destinations is referred to as *matching* and is, in itself, a distributed process, which operates on the distributed management of client interests (advertisements and

subscriptions). Furthermore, the distributed nature of the underlying messaging infrastructure mandates an efficient routing engine that can compute and traverse efficient paths to reach target destinations. We suggest that inefficient approaches to either the calculation of, or routing to, destinations can result in unacceptable network degradations and network *flooding*. Poor solutions to network utilizations lead to buffer overflows, queuing delays, network clogging and other related problems that add up considerably over a period of time. Although multicasting and bandwidth reservation protocols such as RSVP [1] and ST-II [2] can help in better utilizing the network, they require support at the router level. There needs to be a conceited effort to ensure the efficient utilization of networks and networked communal resources.

More importantly, the underlying solution should incorporate sophisticated matching engines needed to provide support for increasingly complex and sophisticated qualifiers, for specifying constraints, that events should satisfy prior to being considered for delivery to applications.

In this paper we explore matching, routing and network utilization issues in the context of our research system NaradaBrokering [3-12], which provides support for centralized, distributed and peer-to-peer (P2P) interactions [13]. NaradaBrokering has been tested in synchronous and asynchronous applications, including as a media server for audio-video conferencing. Depending on the type of interactions routed and the corresponding matching engines supported, the underlying messaging infrastructure/middleware could be viewed either as a distributed light-weight relational or XML database. We also discuss the implications, and include results, pertaining to the different matching engines supported within the NaradaBrokering system.

This paper is organized as follows. In Section 2 we discuss related work in the area. In section 3 we identify the core issues relevant to supporting efficient interactions within the system; sections 4, 5, 6 and 7 elaborate on these core issues of broker topology, organization of client interests/constraints (profiles), routing of events and support for multiple matching engines respectively. Finally in Section 8 we outline conclusions and future work.

2. Related Work

Different systems address the problem of event delivery to relevant clients in different ways. In Elvin [14] network traffic reduction is accomplished through the use of quench expressions, which prevent clients from sending notifications for which there are no consumers. This, however, entails each producer to be aware of all the consumers and their subscriptions. Ref [15] outlines a strategy to convert each subscription in Elvin into a deterministic finite state automaton. This conversion, and the matching solutions, nevertheless can lead to an explosion in the number of states. In Sienna [16] optimization strategies include assembling patterns of notifications

as close as possible to the publishers, while multicasting notifications as close as possible to the subscribers. In Gryphon [17] each broker maintains a list of all subscriptions within the system in a parallel search tree (PST). The PST is annotated with a trit vector encoding link routing information. These annotations are then used at matching time by a server to determine which of its neighbors should receive that event. Approaches for exploiting group based multicast for event delivery is discussed in Ref [18]. The Event Service [19] approach adopted by the OMG is one of establishing channels and subsequently registering suppliers and consumers to the event channels. The approach could entail clients (consumers) to be aware of a large number of event channels. The Notification Service [20] addresses limitations pertaining to the lack of event

filtering capability. However it attempts to preserve all the semantics specified in Event Service while allowing for interoperability between clients from the two services. TAO [21] is a real-time event service that extends the CORBA event service and provides for rate-based

event processing, and efficient filtering and correlation. Unlike Elvin and the OMG Event Service, NaradaBrokering provides decoupled interactions between the interacting clients. Furthermore, the organization of subscriptions and calculation of destinations do not result in explosive

search spaces. As opposed to the Gryphon approach where all nodes store the complete set of subscriptions at every broker node, in NaradaBrokering none of the nodes store all the subscriptions within the system. Also not every broker in NaradaBrokering is involved in the calculation of destinations. This greatly reduces the CPU cycles expended in NaradaBrokering for computing and routing interactions within the system. In some commercial JMS [22] implementations, events that conform to a certain topic are routed to the interested clients with refinement in subtopics being made at the receiving client. This approach could thus expend networkcycles, routing events to clients, where it would ultimately be discarded. JMS systems tend to be single server or limited server solutions. NaradaBrokering is JMS compliant and in Ref [8] we have demonstrated how we can transparently replace single server JMS systems with a distributed solution. In the case of servers that route static content to clients such as Web pages, software downloads etc., some of these servers have their content mirrored on servers at different geographic locations. Clients then access one of these mirrored sites and retrieve information. This can lead to problems pertaining to bandwidth utilization and servicing of requests, if large concentrations of clients access the wrong mirrored-site. In an approach sometimes referred to as active-mirroring, websites powered by EdgeSuite [23] from Akamai, redirect their users to specialized Akamized URLs. Based on the IP address associated with the request the client is then directed to the server farm that is closest to its network point of origin. As network and server loads change clients could be redirected to other servers. In an approach similar to that of Akamai, we are presently incorporating a scheme where clients are directed to the nearest (based on IP address, router hops and communication latencies) available brokers by broker locators available within the system. The JXTA [24] (from juxtaposition) project at Sun Microsystems is a research effort to support large-scale P2P infrastructures. P2P interactions 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. Pastry [25, 26] from Microsoft incorporates a self-stabilizing infrastructure, which provides an efficient location and routing substrate for wide-area P2P applications. Each node in Pastry has a 128-bit ID and Pastry routes messages to nodes whose Node- Id is numerically closest to destination key contained in the message. The JXTA approach results in flooding the peer network, with the range being controlled by the TTL indicators contained in the interactions. The NaradaBrokering scheme selectively deploys links for disseminating interactions. In Ref [7] we have demonstrated that we can route JXTA interactions more efficiently than the JXTA core itself. Distributed Hash Tables (DHTs) have been quite popular in several P2P systems. Here each data object is associated with a key. A lookup service to locate this object returns the IP-address of the node hosting this object. Similar to a traditional hashtable data structure, other operations supported in the DHT include put and get. In P2P overlay networks the nodes are organized based on the content that they possess. Here DHTs are used to locate, distribute, retrieve and manage data in these settings. This scheme provides bounded lookup times. However, P2P overlay networks do not facilitate keyword based searching, the lookups are instead based on identifiers computed by hashing functions such as SHA-1 and are derived from the content encapsulated within the communal resource. Finally, none of the systems that we have described above manage the range of interactions supported within NaradaBrokering. As far as we know we are the only system incorporating Integer, "/" separated Strings, Tag Value, Regular Expressions, XPath and SQL matching engines.

3. Efficient Matching and Routing: Breaking the problem down

The smallest unit of the underlying messaging middleware should be able to intelligently process and route events, while working with multiple underlying communication protocols. We refer to this unit as a *broker*, where we avoid the use of the term *server*s to distinguish it clearly from the application servers that would be among the sources/ sinks to messages generated within the system. Efficient matching and routing of events that builds on solutions to the multiple and sometimes interrelated issues that comprise it. In this section we proceed to outline the four core issues that comprise the problem with subsequent sections discussing each issue in greater detail. First, efficient organization of brokers is important as it forms an important part of the matching and routing solution discussed below. Another competing requirement is the ability of the broker network to adapt to failures that might take place within the system. Inefficient broker organizations can lead to topologies that are susceptible to network partitions upon node failures.

Second, the problem of matching events comprises the related problems of organizing onstraints and efficiently matching events against these constraints to compute destinations. This organization scheme should of course exploit the underlying structure of the broker network.

Third, there is the routing of events to their destinations. This should be done without the need to resort to flooding the broker network, while being able to adapt to the ever changing conditions that exist within a distributed system. Routing decisions, and the routes that need to be taken, are based on the perceived state of the network. A routing solution should be able to factor in network conditions such as failed/clogged/slow links and nodes while making decisions on routes to be taken to reach destinations.

Finally, the specified constraints could be arbitrarily complex, and depending on the application, content and type of the events (and the interactions they encapsulate) that are supported there needs to be multiple matching engines residing within the system.

4. Topology

To address the issues [11] 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. 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. 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" [27,28] where the average communication pathlengths between brokers increase logarithmically with geometric increases in network size, as opposed to exponential increases in uncontrolled settings.

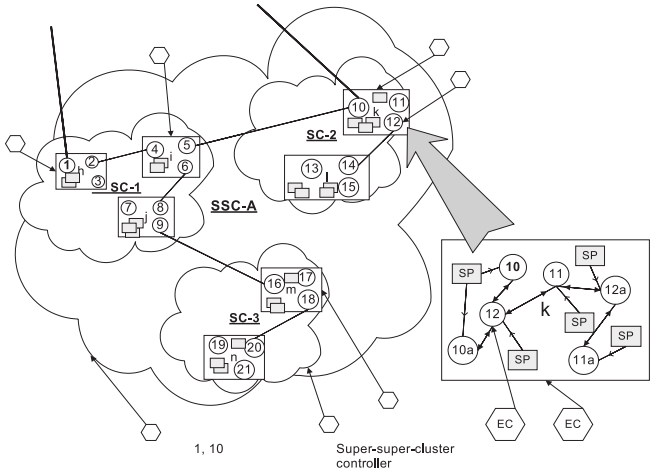


Figure 1: An example of a NaradaBrokering broker network sub-section managing gridlet realms

This distributed cluster architecture allows NaradaBrokering to support large heterogeneous client configurations that scale to arbitrary size. Within every unit (cluster, super-cluster and so on), 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.

4.1 The Broker Network Map (BNM)

A broker needs to be aware of the broker network layout to optimize routing to destinations. However, given the potential size of the broker network, it is impractical for every broker to be aware of the complete broker network inter-connection scheme. What is required is an abstract view of the broker network, while still being able to ensure the calculation of optimal paths for communication within the system. This information is encapsulated within the BNM. The information encapsulated within the BNM provides information regarding the inter-connections between the brokers in the cluster that it is a part of, the interconnections between the clusters within the super-cluster that it belongs to and so on. The BNM maintained at each broker node is different, while still providing a consistent view of the system interconnections.

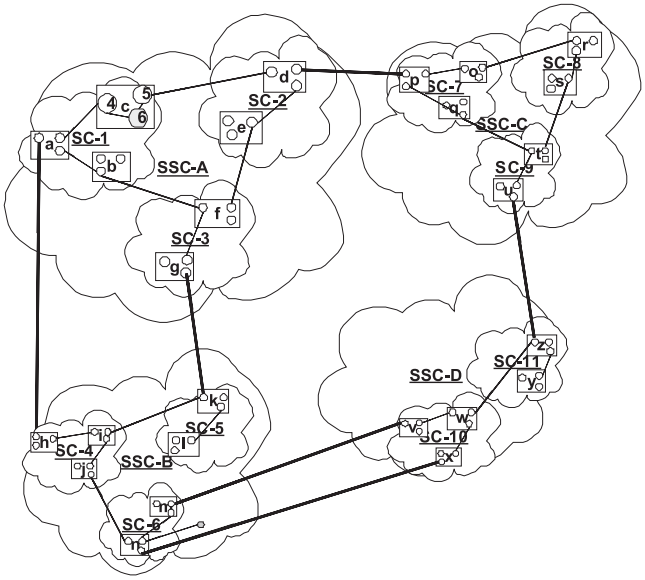


Figure 2: An example broker network

Changes to the broker network fabric are propagated only to those brokers that have their broker network view altered. BNMs at each node need to be updated in response to the receipt of information pertaining to the creation of connections between brokers/units. Dissemination constraints are imposed on the propagation of connection information outside a given unit. For example information regarding connections between brokers within a cluster should not be propagated outside the cluster. Connection information is also modified as it is being propagated through certain sections of the broker network. Thus, in Figure 2 the connection between **SC-2** and **SC-1** in **SSC-A**, is disseminated as one between node **5** and **SC-2**. When this information is received at **4**, it is sent over as a connection between the cluster **c** and **SC-2**. When

the connection between cluster **c** and **SC-2** is sent over the cluster gateway to cluster **b**, the information is not updated. Conforming to the dissemination constraints, the super cluster connection (**SC-1,SC-2**) information is disseminated only within the super-super-cluster **SSC-A** and is not sent over the super-super-cluster gateway available within the cluster **a** in **SC-1** and cluster **g** in **SC-3**.

Figure 3 depicts the BNM at node **6**. We augment the BNM hosted at individual brokers to reflect the cost associated with traversal over connections, for example intra-cluster communications are faster than inter-cluster communications. This cost can be dynamically updated to reflect changes in link behavior with the passage of time. The BNM can now be used not only to compute valid paths but also for computing shortest paths.

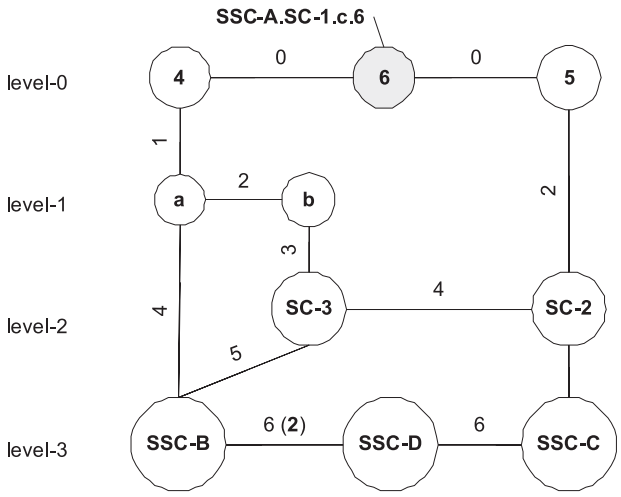


Figure 3: The Broker Network Map at node 6

5. Organization and Propagation of Profiles

Profiles signify an interest in events conforming to a certain template. Profiles also include a constraint that events need to satisfy, before being considered for routing to a client. This is generally referred to as a subscription. Constraint complexity can vary from character-string based topic matching to a sophisticated SQL or XPath query. Individual profiles can also include information pertaining to the device type – CPU capability, and security related information that would sometimes be needed for the matching process. Every profile has a unique ID associated

with it which plays an important role in the management – addition, removal and organization – of profiles. Profile organizations and propagations are inter-related issues, which need to exploit the topology, and the organization of units and controllers within the system. The organization of profiles needs to be such that it reduces the number of matching steps that need to be performed. Propagations need to be sophisticated enough to ensure that profiles are propagated only to relevant nodes within the system. Every profile has an associated destination, which is updated depending on its propagation within the system. A profile is propagated to unit controllers, and the destination associated with the profile during its storage at the unit controller is that of the sub-unit controller that propagated it. The hierarchical propagation of profiles – resulting in a broker maintaining profiles of all attached clients, cluster controllers maintaining profiles of all brokers within that cluster and so on – ensures that when an event is routed to a unit, there is at least one final destination within that unit. The scheme also ensures that a matching event is routed

to every valid destination without exception. Thus, in Figure 1, super-super-cluster controller nodes **1** and **10** keep track of all profiles 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 propagated by nodes **19**,**20**, **21** in cluster **n**. Since a unit controller operates and communicates only with sub-unit destinations, all profiles are stored at the controllers as if they originated at specific sub-units. Thus, for a profile 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**.

Another factor that is equally important is the removal of profiles from propagation trees. This is done sometimes based on a explicit removal propagation initiated by a client and also depending on the loss of connection to a certain client. In either case the issue is an important

one to ensure that network and CPU cycles are not expended while trying to reach destinations that are not truly interested in the event in the first place. Finally, in this scheme, for system wide dissemination every event needs to arrive at, at least one super-super-cluster controller, within every super-super-cluster. The advantage of this scheme is that no node maintains the complete

list of client profiles in the system. This could result in a super-super-cluster being overloaded during high volume interactions. This problem can be alleviated considerably by having multiple super-super-cluster controllers within any given super-super-cluster.

6. Routing Events to Destinations

Event routing is the process of disseminating events to relevant clients. This includes matching the content, computing the destinations and routing the content along to its relevant destinations by determining the next broker node that the event must be relayed to. As an event flows through the system, via unit controllers, the associated event distribution trace is modified to snapshot the event's dissemination within the broker network. These routing traces indicate – and can be used to verify – an event's dissemination within various parts of the broker network. Routing decisions are made on the basis of this trace information and the computed destinations.

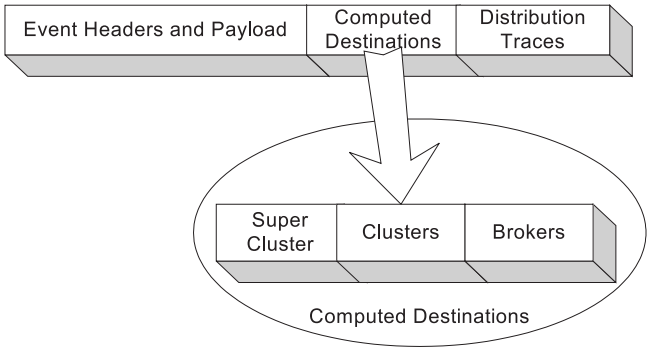


Figure 4: Event destinations and traces

The matching process at a unit-controller computes sub-unit destinations, which are valid only within that unit. Figure 4 shows the destinations associated with an event in a system comprising of super-super-clusters. From the stored BNMs at each node, individual unit-controllers compute the best routes to reach units contained in the destinations. When an event arrives at a unit-controller, prior to being sent over the link to another unit, the sub-unit destinations associated with the event is invalidated. Thus, broker destinations computed by a cluster controller are valid only within that cluster and are cleared prior to routing the event to another cluster.

Before an event is sent over a link to another unit, unit-controllers analyze the trace information to ensure that the event is not routed to a unit, where the event has already been routed. At every node the best hops to reach the destinations are computed. Nodes and links that have not been failure suspected are the only entities that can be part of the shortest path. Thus, at every node the best decision is taken based on the current state of the network fabric.

6.2 Communication overheads

We now present some results from NaradaBrokering's transport framework. The results give an idea of the overheads involved in communications. NaradaBrokering supports multiple transport protocols such as TCP, UDP, Multicast, SSL, HTTP and HTTPS. The results we report here are for TCP based communications. The graph in Figure 5 depicts the mean transit delays, and the accompanying standard deviations, for NaradaBrokering messages traversing through 2 hops with a single broker in the path from the sender of the message to the receiver. For each test case the message payload was varied. The transit delay plotted is the average of the 50 messages that were published for each payload. The sender/receiver pair along with every broker involved in the test cases were hosted on different physical machines (Pentium-3, 1 GHz, 256 MB RAM). The machines reside on a 100 Mbps LAN. The run-time environment for all the processes is JRE-1.3 build Blackdown-1.3.1, Red Hat Linux 7.3.

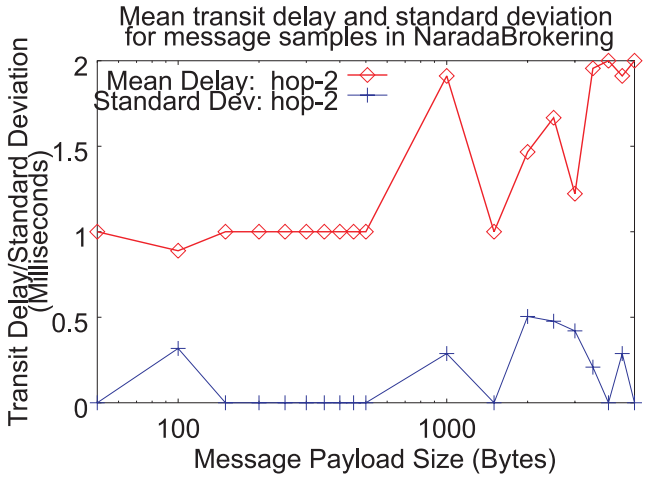


Figure 5: Mean delay and standard deviation for varying payload sizes

7. The Matching Engine

In this section we discuss the matching process and the assortment of matching engines residing in NaradaBrokering. The matching engine is responsible for computing destinations associated with an event based on the profiles available at a node. Depending on the type of applications, standards, events and subscriptions that need to be supported there would be multiple matching engines residing within every processing broker node.

For several reasons we limit the number of sub-units within a unit to 32. 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. The implications of the representation, and the upper-bound on sub-units, are even more powerful in the context of computing destinations efficiently. Individual profiles have destinations associated with them. A unit-controller maintains profiles with sub-unit destinations. The number of profiles that are maintained at a controller progressively increases depending on whether the controller in question is a broker, cluster-controller, super-cluster

controller and so on. A unit-controller computes sub-unit destinations, and the destinations that are associated with the stored profiles are also sub-unit 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. 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.

7.1 The Assortment of matching engines

We now proceed to discuss individual matching engines that reside within NaradaBrokering.

7.1.1 String based matching

This matching is based upon the generalized String topic-based publish/subscribe paradigm. Events issued provide information regarding the topic that they were issued to. Client profiles include a subscription to a topic. If the topic contained in the event is the same as the topic contained in the profile, the event is said to match the profile. This is a powerful model and several sophisticated applications can be built using this generalized publish/subscribe model.

Some systems incorporate an approach to topic matching where a subscription to a topic, say Sports, translates into subscriptions to all sub-topics, say Sports/NBA, Sports/Soccer/UEFA. This approach is not supported in NaradaBrokering due to constraints imposed by the message-based security scheme [29].

7.1.2 String based matched coupled with SQL-like queries on properties

Events (or messages) may also include properties, which are used to further describe the content contained in the event's payload. Clients can thus also incorporate a second level of refinement for the events they are interested in.

This two layer refinement scheme has the advantage that the first constraint, which is identical to the string-based topic matching scenario that we outlined earlier, substantially reduces the number of events on which the second refinement needs to be applied. This is important since the second level of refinement is far more complex and CPUintensive than the first one.

The JMS specification incorporates this strategy, with the refinement syntax being based on a subset of the SQL92 conditional expression syntax. If the value of a refinement is an empty string, it indicates that there no refinement is specified and the case reduces to the topic based publish/subscribe outlined above.

7.1.3 Topics that are based on tag=value pairs

This matching engine incorporated into NaradaBrokering, is based on the equality-based generalized matching algorithm presented in [30]. Topics in this case comprise of equality constraints imposed on a set of successive attributes as a sequence of "," separated <*tag, value*> pairs. The constraint in this case is the specification of a *value* that a particular attribute (*tag*) can take. Also allowed is the weakest constraint, denoted *, which encompasses all values. In this case subscribing to a topic Make=Ford,Model=*,Color=Red matches events with topic Make=Ford,Model=Taurus,Color=Red and Make=Ford,Model=Mustang,Color=Red. Based on the number of <tag,values> specified and the tags with specified * constraints, the complexity of the matching process increases.

7.1.4 Integer based matching

The Integer based topic matching is used in NaradaBrokering primarily by the audio/video conferencing framework to enable real time communications [10].

7.1.5 XML based matching with XPath queries

NaradaBrokering also incorporates support for XPath based specification of constraints on XML events. XPath [31] 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.

7.1.6 Regular expressions based matching

More recently, NaradaBrokering incorporates support for regular expressions based matching. Regular expressions are a powerful mechanism for incorporating complex constraints on text based content. Regular expressions of the form [Tt]he [Qq]uick [Bb]rown [Ff]ox [Jj]umps can be satisfied by a topic of the form "The quick brown fox jumps …".

7.2 Profiling the Matching Engines

We now provide some results pertaining to the matching engines that were outlined in the earlier section. These results (Figures 6 through 11) are for stand-alone processes, where we computed the matching times as a function of the number of subscriptions maintained. In each case, an event is matched to retrieve every matching subscription. For every matching engine, the number of subscriptions is varied from 10,000 to 100,000. The results were measured on a machine (1GHz,256MB RAM) running the process in a Java-1.4 Sun VM with a high-resolution timer for computing delays.

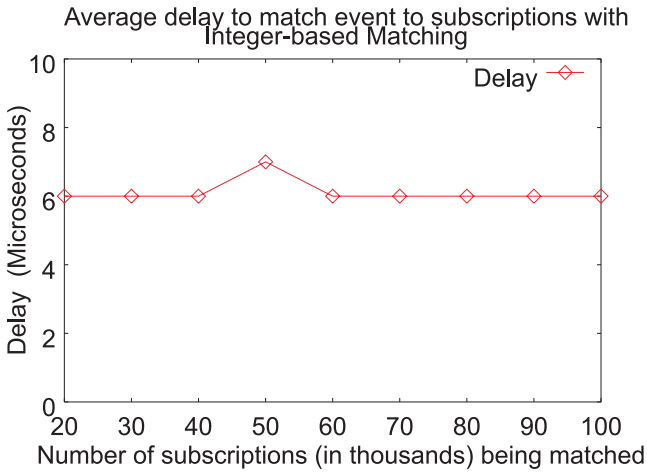


Figure 6: Plots for Integer matching

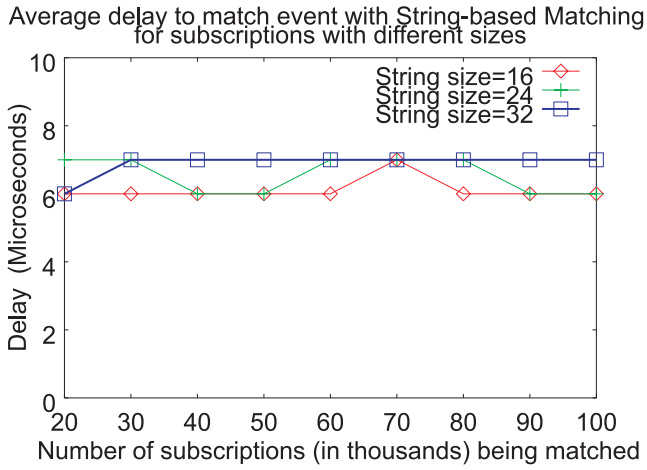


Figure 7: Plots for String based matching

The richer the constraints, the greater the CPU-cost associated with the matching process. As can be seen the average delays for matching increases progressively (Integer to String to Tag-Value to SQL to XPath in that order) as the complexity of the matching increases. For String based matching, as depicted in Figure 7, the average delay for matching subscriptions generally increases as the size of the topic String increases. The increase in delays for matching as the topic String size doubled from 16 to 32 was in the range of 1 microsecond. Figure 8 depicts the costs associated with <tag,value> based matching. As can be seen the costs associated with this style of matching is higher than the String based style. We also noted that the costs did not vary as the number of <tag,value> pairs associated with individual subscriptions increased from 5 to 25. The results in Figures 6-8 demonstrate that it is feasible to have real time interactions that are based on the corresponding constraints.

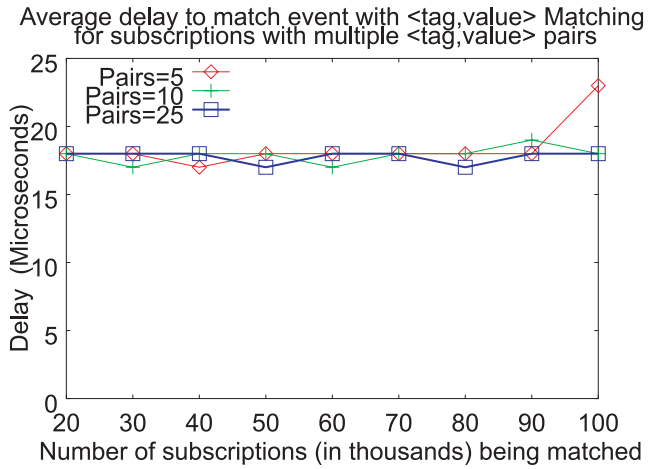


Figure 8: Plots for <tag,value> based matching

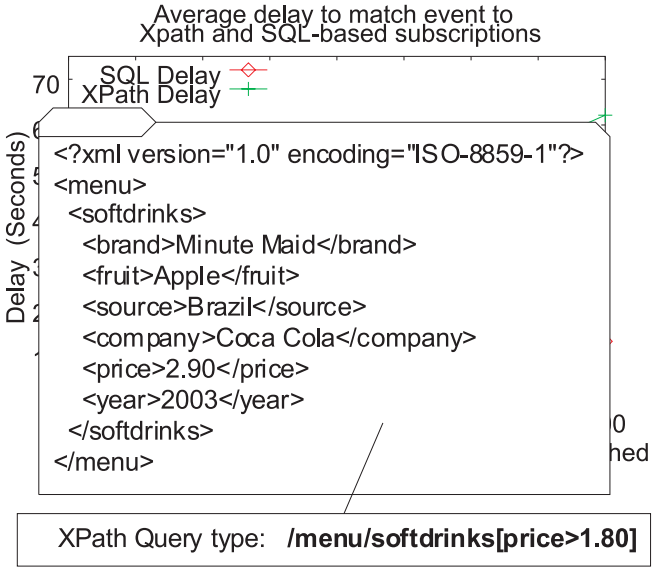
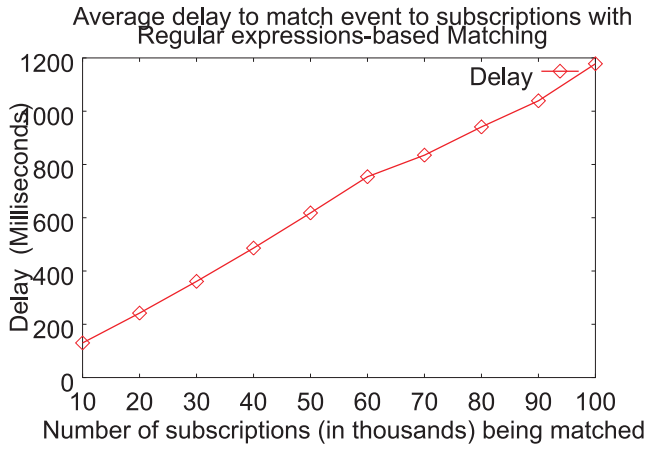


Figure 9: Plots for SQL and XPath based matching

Figure 9 contrasts the costs involved in matching JMS events to stored SQL-92 based selectors on the properties contained within the JMS message and XML events to stored XPath conforming constraints. Of course these costs can vary significantly depending on the type of the query. For our experiments we used XPath and SQL queries,



which we felt were comparable. The cost of a single matching operation involving an XML event and an accompanying XPath query is around 3 milliseconds.

Figure 10: Regular expressions based matching

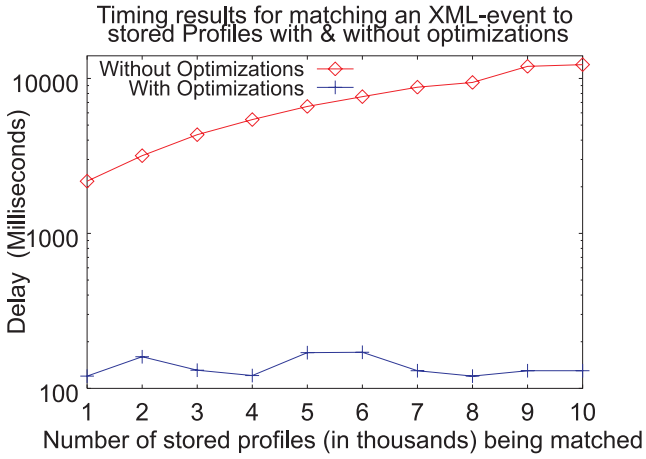


Figure 11: Matching XPath profiles

Figure 10 depicts the costs involved in matching an event to regular expression based constraints. The subscriptions stored were in the following format: [Tt]he [Qq]uick [Bb]rown [Ff]ox [Jj]umps while the topic was of the form “The quick brown fox jumps upon the lazy dog”. The matching costs varied from 130 milliseconds for 10000 subscriptions to 1178 milliseconds for 100000 subscriptions. Next, we proceeded to study the effects of the destination optimization strategies, which we discussed in section 7,in the context of XPath profiles. We first perform the matching (un-optimized) on a set of XPath profiles. The Xpath profiles in this case are then evenly distributed over 32 different destinations. Figure 11 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 12 outlines the XML type for the stored events, and the type of XPath query used in our experiments. Though the results depicted here are for XPath profiles we expect optimizations to have a similar effect on SQL based profiles too.

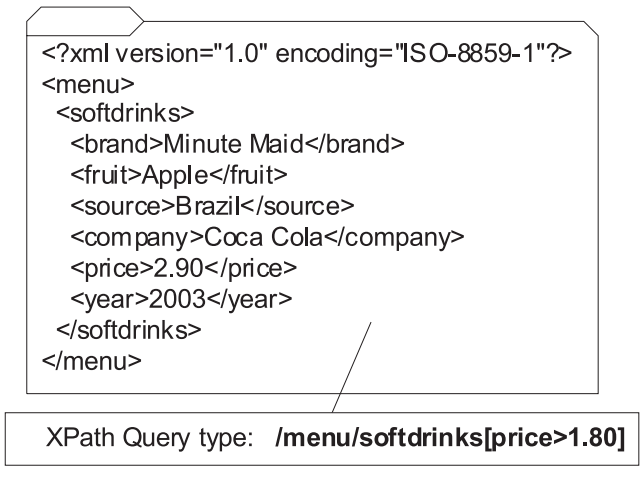


Figure 12: The XML event and XPath query type

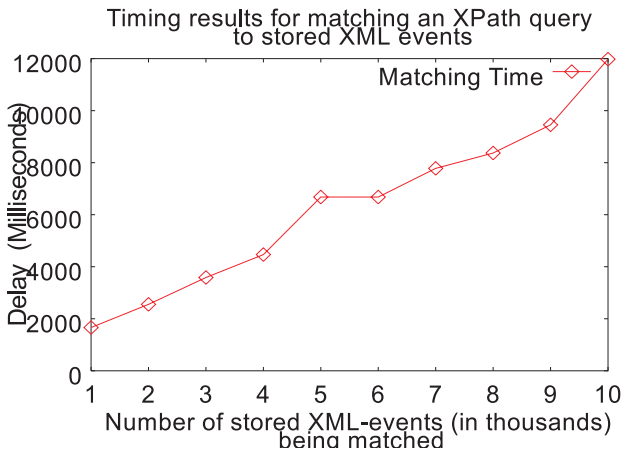


Figure 13: Matching an XPath query to stored XML events

7.3 Implications of query based matching engines

The Query-based engines are suitable for discovery based services. While providing support for profiles with SQLlike query based refinements and XPath query based profiles, the system can be viewed as a lightweight, distributed relational and XML database respectively. This is the case, since as far as the end-user is concerned, the matched event might as well have been stored in a database (relational or XML, as the case might be) and the results returned (matching events) would not have been different. Clients in the system can advertise their services in an XML schema or a schema that can be queried by an SQL query. These advertisements would be stored in the same way that the profiles are stored within the system. Events propagated by interested clients would essentially be either XPath or SQL-like queries. These events would then be matched against the stored advertisements with the matching ones being routed back to the initiating client. The query events can specify the realms within which the query's propagation might take place, thus allowing individual entities to control how localized their services can be.

Figure 13 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 or stored XML events that need to be included in the query response. The stored XML events and the issued Xpath query are of the type depicted in Figure 12. For most discovery related operations, similar to those initiated in P2P systems, these numbers indicate adequate performance.

8. Conclusions and Future work

The matching problem is a sufficiently difficult and important problem, which needs to be addressed within the messaging infrastructure that supports the applications, and accompanying interactions, between entities. The problem will continue to evolve as entities continue to interact in increasingly complex ways. In this paper we discussed issues, and strategies, to support efficient matching of events. Based on the kind of applications that the system is trying to support, optimized engines that employ optimistic delivery techniques (based on routing behavior of past events) could also be deployed.

P2P search mechanisms employ strategies different from those discussed above. Combining P2P search mechanisms initiated by peers on the edge of the network with the schemes outlined in earlier sections, provides interesting approaches to resource management that would be of considerable interest. Managing interactions between Web/Grid services generated dynamically when complex tasks are initiated is another area of research. Finally, incorporating some of the security related information (SAML [32] style authorizations) into the profiles themselves would allow us to be even more selective of the events being routed to entities.

References

1. Zhang, L. et al.(1994). ReSource ReserVation Protocol (RSVP) – Functional Specification", Internet Draft.

2. Topolcic, C (1990). Experimental Internet Stream Protocol: Version 2 (ST-II)", Internet RFC 1190.

3. The NaradaBrokering System http://www.naradabrokering.org

4. Fox, Geoffrey, Pallickara, Shrideep (2003). NaradaBrokering: An Event Based Infrastructure for Building Scaleable Durable Peer-to-Peer Grids. Chapter 22 of "Grid Computing: Making the Global Infrastructure a Reality". Published by John Wiley, West Sussex, England. ISBN 0-470-85319-0.

5. Pallickara, Shrideep, Fox, Geoffrey (2003). NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware.

6. Pallickara, Shrideep, Fox, Geoffrey (2004). On the Matching Of Events in Distributed Brokering Systems. (To appear) Proceedings of IEEE ITCC Conference on Information Technology. April.

7. Fox, Geoffrey, Pallickara, Shrideep, Xi, Rao (2002). A Scaleable Event Infrastructure for Peer to Peer Grids. Proceedings of ACM Java Grande ISCOPE Conference 2002. Seattle, Washington..

8. Fox, Geoffrey, Pallickara, Shrideep (2002). JMS Compliance in the NaradaBrokering System. Proceedings of the International Conference on Internet Computing (IC-02). June 2002. 391-402.

9. Pallickara, Shrideep, Fox, Geoffrey. A Scheme for Reliable Delivery of Events in Distributed Middleware Systems. (To appear) Proceedings of the IEEE International Conference on Autonomic Computing. New York, NY.

10. Bulut et al (2002). Integration of NaradaBrokering and Audio/Video Conferencing as a Web Service. Proceedings of the IASTED International Conference on Communications, Internet, and Information Technology.

11. Fox, Geoffrey, Pallickara, Shrideep (2001). An Approach to High Performance Distributed Web Brokering. *ACM Ubiquity* 2:38.

12. Fox, Geoffrey, Pallickara, Shrideep . An Event Service to Support Grid Computational Environments. *Journal of Concurrency and Computation: Practice & Experience*. 14(13-15) 1097-1129.

13. Oram, Andy 92001). Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. Edited by Andy. O'Rielly Press, CA.

14. Segall, Bill, Arnold, David. (1997). Elvin has left the building: A publish/subscribe noti.cation service with quenching. *In:* Proceedings AUUG97, pages 243–255, Canberra, Australia, September 1997.

15. Segall, Bill, Arnold, David, Boot, Julian, Henderson, Michael, Phelps, Ted.(2000) Content based routing with elvin4. *In:* Proceedings AUUG2K, Canberra, Australia, June 2000.

16. Carzaniga, Antonio,. Rosenblum, David S, Wolf, Alexander L. (2000) Achieving scalability and expressiveness in an internetscale event notification service. In Proceedings of the 19th ACM Symposium on Principles of Distributed Computing, 219–227, Portland OR, USA.

17. Banavar, G et al (1999). An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In Proceedings of the IEEE International Conference on Distributed Computing Systems, Austin, Texas.

18. Opyrchal, Lukasz et. al (2000). Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. Middleware: 185-207

19. The Object Management Group (OMG). OMG's CORBA Event Service. Available from http://www.omg.org/

20. The Object Management Group (OMG). OMG's CORBA Notification Service. Available from http://www.omg.org/

21. Harrison, T.H, Levine, D.L, Schmidt, D.C (1997). The design and

performance of a real-time CORBA object event service. Proceedings of the OOPSLA'97. Atlanta, GA.

22. Happner, Mark Burridge, , Rich, Sharma, Rahul (2000). Java Message Service Specification. Sun Microsystems. http://java.sun.com/products/jms

23. Akamai Corporation. EdgeSuite: Content Delivery Services . Technical report, URL: http://www.akamai.com/

24. Sun Microsystems. The JXTA Project and Peer-to-Peer Technology http://www.jxta.org

25. Rowstron, Antony, Druschel, Peter (2001). Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. Proceedings of Middleware.

26. Squirrel (2002) A decentralized peer-to-peer web cache. ACM PODC.

27. Watts, D.J, Strogatz, S.H (1998). Collective Dynamics of Small-World Networks. *Nature*. 393:440.

28. Albert, R, Jeong, H, Barabasi, A (1999). Diameter of the World Wide Web, *Nature* 401:130.

29. Pallickara, Shrideep et al (1999). A Security Framework for Distributed Brokering Systems. (Under Review).30. Marcos Aguilera et al. Matching events in a content-based subscription system. In Proceedings of the 18th ACM Symposium on Principles of Distributed Computing.

30. XML Path Language (XPath). Version 1.0. W3C Recommendation. Available from http://www.w3.org/TR/xpath

31. Hallam-Baker, P, Maler,E. Assertions and Protocol for the OASIS Security Assertion Markup Language. P. and E. Maler, eds. Available from http://www.oasis-open.org/committees/security/docs/ cs-sstc-core-01.pdf.