On the Matching of Events in Distributed Brokering Systems

Shrideep Pallickara and Geoffrey Fox
{spallick,gcf}@indiana.edu
Community Grid Computing Laboratory, Indiana University

1.0 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. Forrester Research, a technology consultancy, predicts [1] that 14 billion such devices will be connected to the Internet by 2010. 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. Clients have transient connection semantics and are themselves originators of voluminous content. 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.

The processing and servicing of events is in itself a distributed problem that involves several nodes and the links that connect them. 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). The distributed nature of the underlying messaging infrastructure also mandates an efficient routing engine.

We suggest that inefficient approaches to either the calculation of, or routing to, destinations can result in unacceptable network degradations. Solutions that unintelligently forward all events to all participating nodes of the underlying infrastructure are said to be flooding the network. Such solutions result in network and CPU cycles being expended in the processing of events that should not have been routed to a node in the first place. 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. Straitjacketing clients by allowing a fixed set of accesses, preventing certain types of interactions, limiting the type of content that is routed to them or even restricting accesses to a fixed number of clients is not the solution. Although multicasting and bandwidth reservation protocols such as RSVP [2] and ST-II [3] can help in better utilizing the network, they require support at the router level. More conceited effort is needed at higher levels. There needs to be a conceited effort to ensure the efficient utilization of networks and networked resources. Inefficient utilizations of the communal resource, in this case the underlying networks, degrades the value of the resource for other applications/infrastructures that utilize this resource. The onus of providing an efficient service that takes network conditions into account rests with the middleware.

More importantly, the underlying solution should support sophisticated search/discovery and matching of events while allowing arbitrarily complex applications to be built upon these solutions. Resident matching engines need 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 prototype NaradaBrokering [4-12], 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 could be viewed either as a distributed light-weight relational or XML database. We discuss the implications, and include results, pertaining to the different matching engines supported within the NaradaBrokering system.

2.0 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. [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 [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 and the inability to configure support for different qualities of service. 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.

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 network-cycles, routing events to clients, where it would ultimately be discarded.

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 the network load and server loads change clients could be redirected to other servers.

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 and also by the peer traces that eliminate the continuous echoing problem caused by traces in peer connectivity. Pastry [25] 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. Squirrel [26] uses this to implement a distributed cache where resources are cached at the node that is numerically closest to the hash value of the resource.

3.0 Efficient Matching and Routing: Breaking the problem down

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 these issues with subsequent sections discussing each issue in more detail. The smallest unit of the underlying messaging infrastructure 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 organization of brokers is important as it forms an important part of the solution. Routing algorithms, when they are presented with a set of destinations, need to compute paths to reach those destinations. Efficient solutions

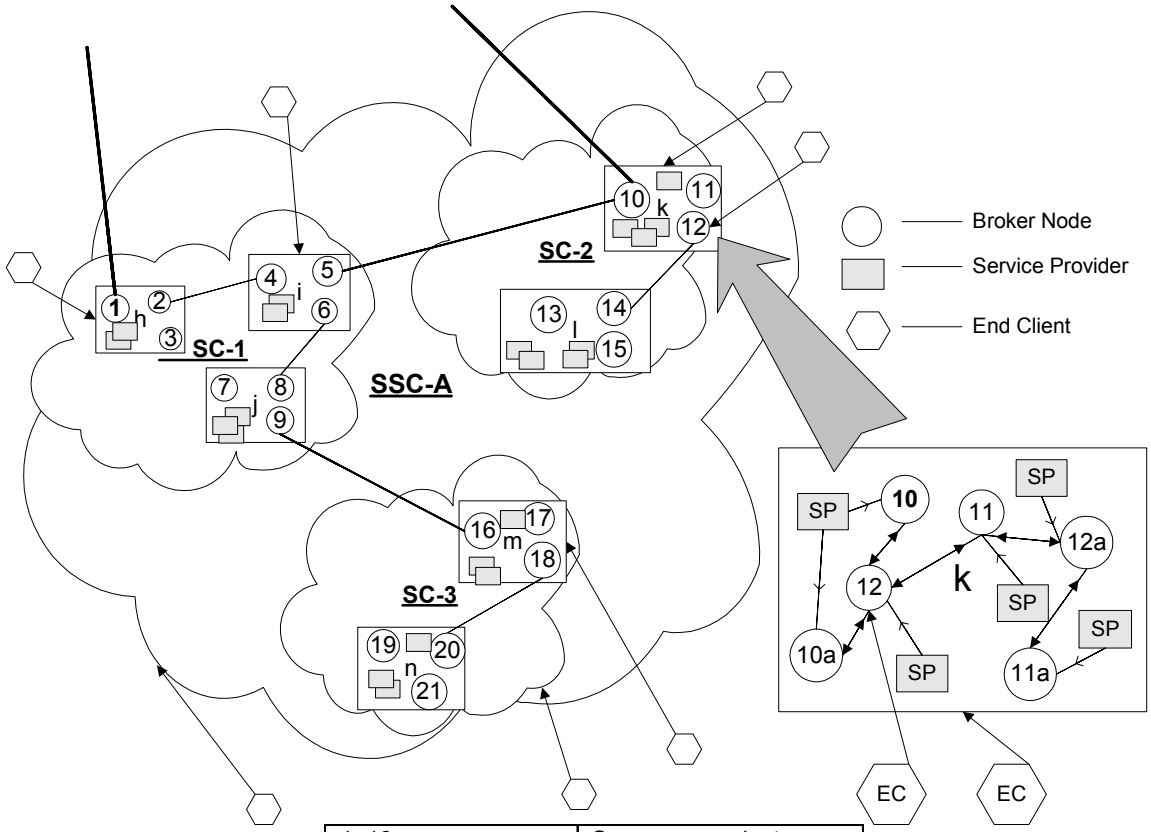
usually operate on broker network maps (BNMs), stored at every broker node, which essentially provide a snapshot of the underlying infrastructure. Another competing requirement is the ability of the broker network to adapt to failures that might take place within the system. Inefficient broker organizations can lead to topologies that are susceptible to network partitions upon node failures.

The problem of matching events comprises the related problems of organizing constraints and efficiently matching events against these constraints to compute destinations. 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.

Then, 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 links and nodes, clogged links, slow nodes while making decisions on routes to be taken to reach destinations. Another important factor is the ability to deploy the right transports for the most efficient communications for e.g. using UDP while routing video packets.

4.0 Topology

To address the issues [11] of scaling, load balancing and failure resiliency, NaradaBrokering is implemented on a network of cooperating brokers.



1, 10	Super-super-cluster controller
5, 9, 10, 16	Super-cluster controller
2,4, 6,8, 12,14,18,20	Cluster controller

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

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. 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 any 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.

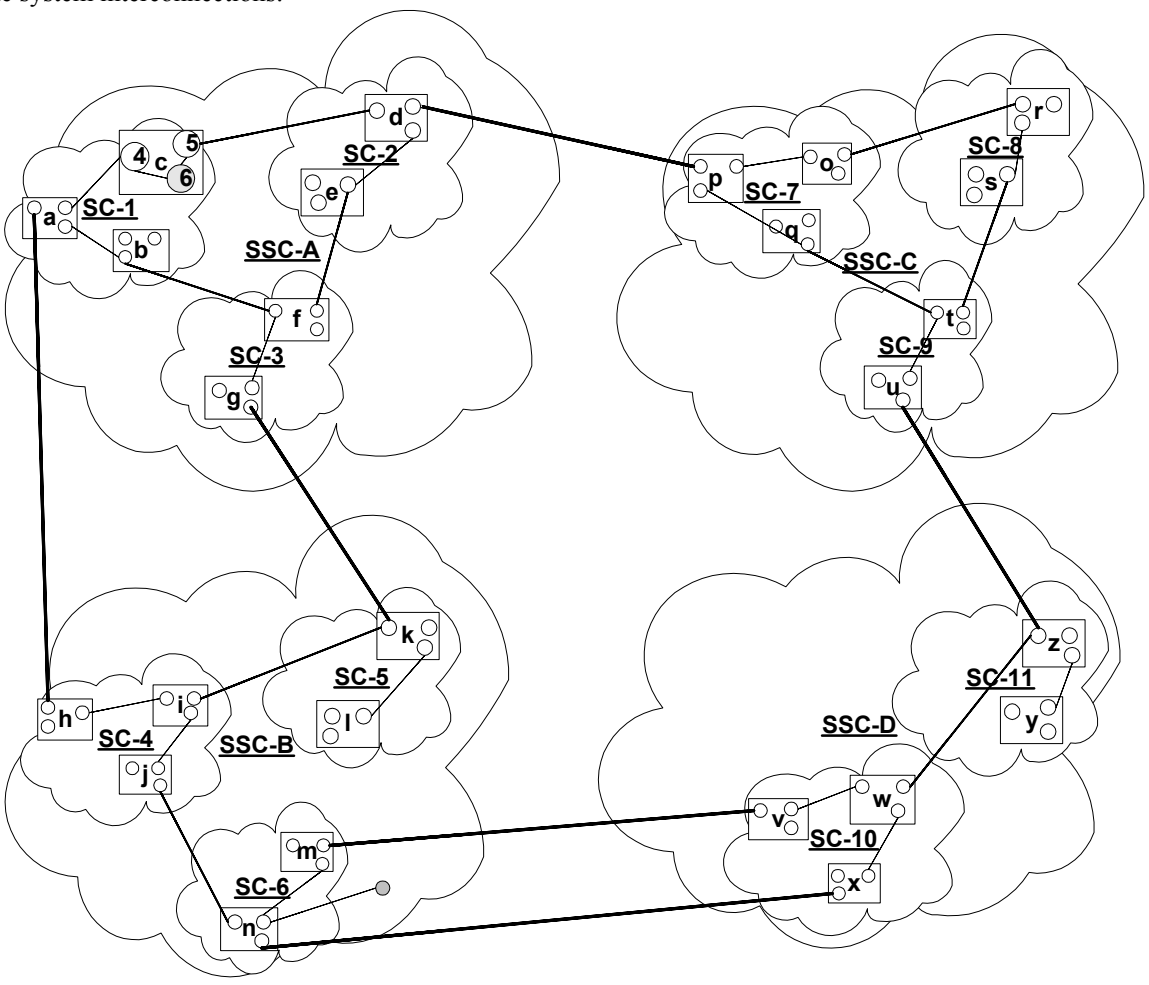


Figure 2: An example broker network

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 within a cluster should not be propagated outside the cluster. This 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**.

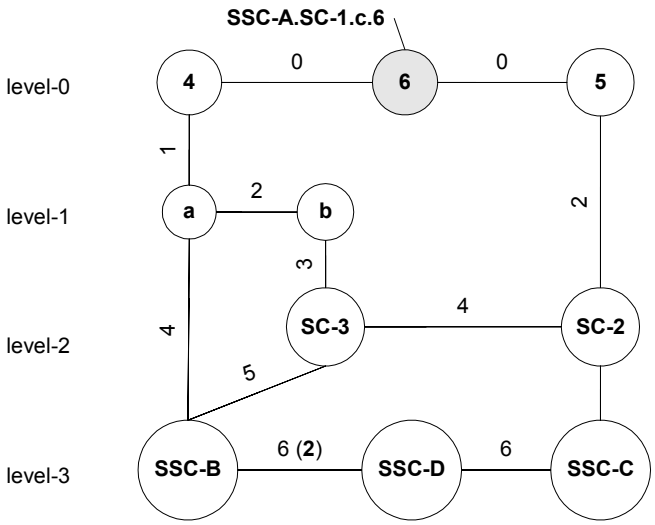


Figure 3: The Broker Network Map at node 6

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.

5.0 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 – processing 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 and removal – 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.

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. While dealing with clients with transient connection semantics, a loss in connection could be detected and the profiles associated with the client need to be scheduled for removal. In the case of TCP, a loss in socket connection can be easily detected. In the case of UDP communications, a ping/reply scheme can be incorporated to detect the client's digital presence.

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. Thus, when a profile is propagated by a client to the broker it is connected to, the broker propagates the profile to its cluster controller with the broker as the destination. The cluster controller in turn propagates the profile to the super-cluster controller with itself as the destination. By controlling the propagations of profiles, a client can control how localized its matched events would be.

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.

6.0 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 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.

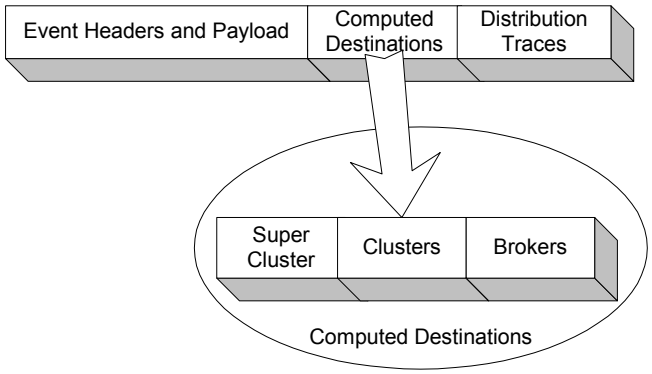


Figure 4: Event destinations and traces

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.

7.0 Experimental Results

Figures 6 and 7 illustrate some results from our initial research where we studied the message delivery time as a function of load. The results are from a system comprising 22 broker processes and 102 clients in the topology outlined in Figure 5. Each broker node process is hosted on 1 physical Sun SPARC Ultra-5 machine (128 MB RAM, 333 MHz), with no SPARC Ultra-5 machine hosting two or more broker node processes. The publisher and the *measuring* subscriber reside on the same SPARC Ultra-5 machine. In addition to this there are 100 subscribing client processes, with 5 client processes attached to every other broker node (broker nodes **22** and **21** do not have

any other clients besides the publisher and measuring subscriber respectively) within the system. The 100 client node processes all reside on a SPARC Ultra-60 (512 MB RAM, 360 MHz) machine. The run-time environment for all the broker node and client processes is Solaris JVM (JDK 1.2.1, native threads, JIT). We measure the latencies at the client under varying conditions of publish rates, event sizes and matching rates. In most systems where events are continually generated, a "typical" client is generally interested in only a small subset of these events. This behavior is captured in the matching rate for a given client. Varying the matching rates allows us to perform measurements under conditions of varying selectivity. The 100% case corresponds to systems that would flood the broker network. In systems that resort to flooding (routing a message to every router node) the system performance does not vary with changes in the match rate. Furthermore, in most cases a given message would only be routed to a small set of targeted client nodes. In NaradaBrokering, as the results clearly demonstrate, the system performance improves significantly with increasing selectivity from subscribers. We also found that the distributed network scaled well with adequate latency unless the system became saturated at very high publish rates.

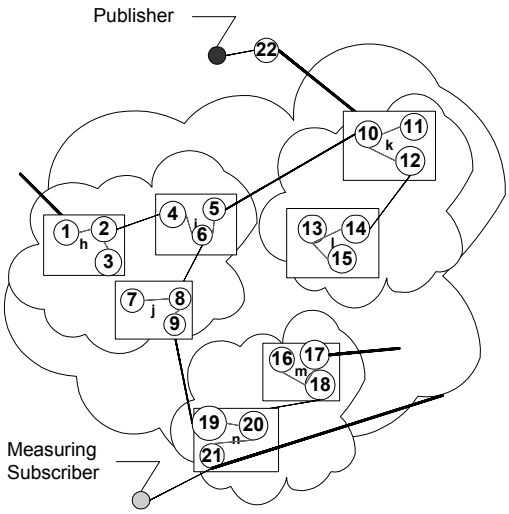


Figure 5: The NaradaBrokering Test Topology

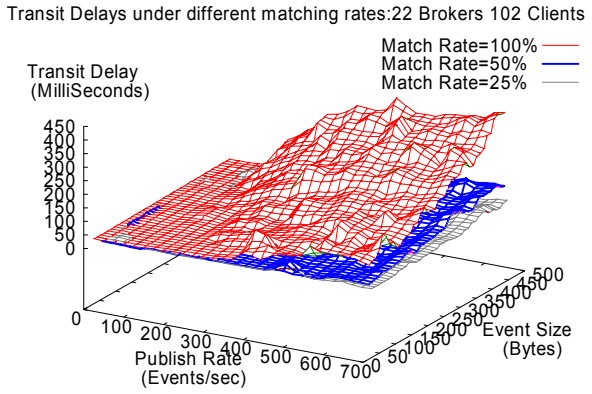


Figure 6: NaradaBrokering Performance at match rates of 100%, 50% and 25%

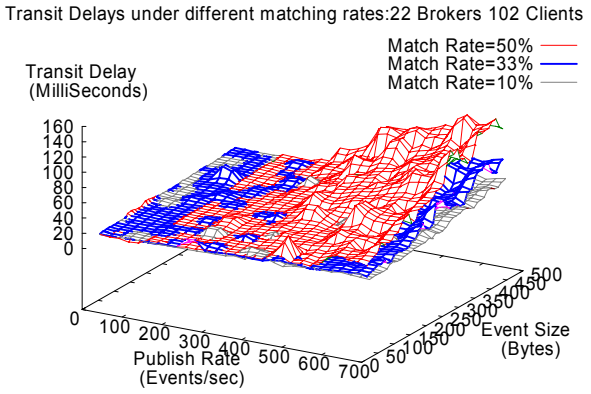


Figure 7: NaradaBrokering Performance at match rates of 50%, 33% and 10%

8.0 The Matching Engine

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). This also 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.

8.1 The Assortment of matching engines

In this section we go discuss the various matching engines residing in NaradaBrokering.

8.1.1 String based matching

This matching is based upon the generalized 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 a hierarchical 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 where the message-based security scheme [29] would have resulted in overheads pertaining to topic-key distributions when a new sub-topic is created. Compromise of topic keys at one user would then have resulted in key invalidations of all sub-topic keys that were routed to the user.

8.1.2 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 CPU-intensive 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.

8.1.3 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 also `Make=Ford,Model=Mustang,Color=Red`. Depending on the number of <tag,values> specified and the tag (and number of tags) at which the constraint * is specified the complexity of the matching process increases.

8.1.4 Integer based matching

In one of our earliest attempts [10] we provided support for audio/video conferencing by encapsulating RTP packets in events, with the topic identifier being a String based meeting-ID. This had a couple of drawbacks. First, String

based representations added a byte-per-character of the meeting-ID in the serialized representation of the event. Second, we sought to reduce the overhead associated with string based matching. The solutions to these issues was to have a unique integer as the meeting-ID, the serialized representation of which was always 4 bytes while significantly reducing the corresponding matching times. The time saving based on this approach accumulate over a period of time resulting in substantial gains in response times as well as network utilizations.

8.1.5 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.

8.2 Profiling the Matching Engines

We now provide some results pertaining to the matching engines that were outlined in the earlier section. These results (Figures 8 through 12) 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. The richer the constraints, the greater the CPU-cost associated with the matching process.

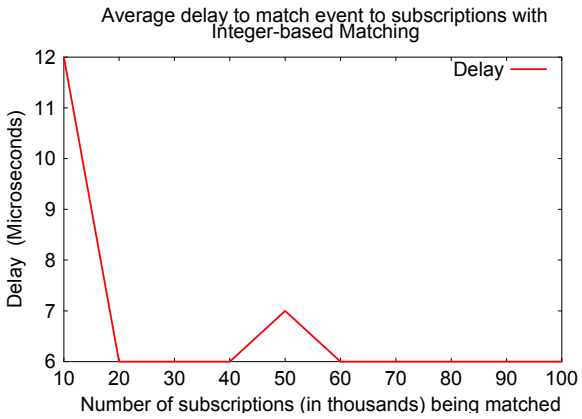


Figure 8: Plots for Integer matching

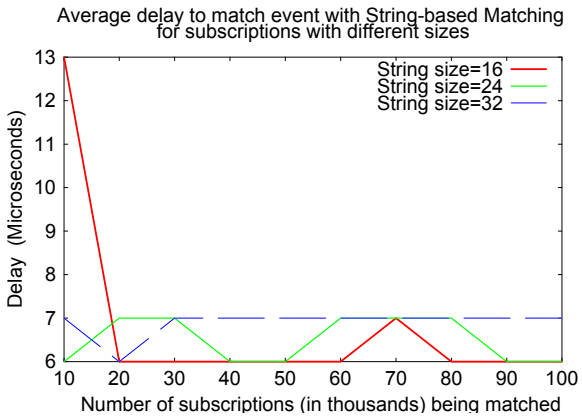


Figure 9: Plot for String based topic matching

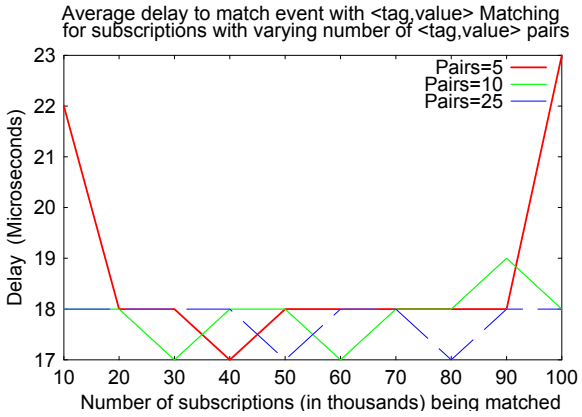


Figure 10: Plots for tag=value based Matching

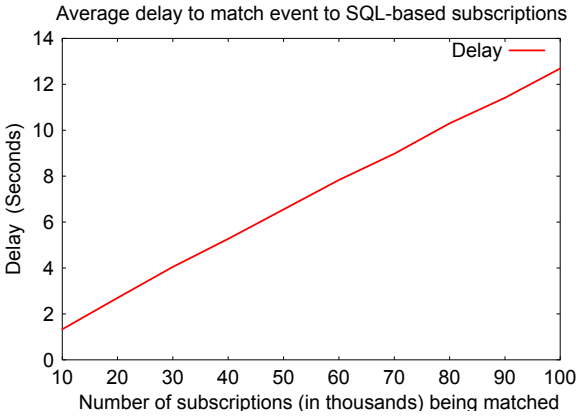


Figure 11: Plots for SQL-query based matching

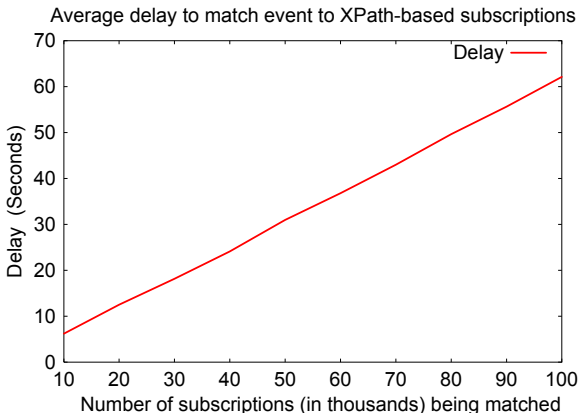


Figure 12: Plots for XPath query based matching

8.3 Implications of query based matching engines

The Query-based engines are suitable for discovery based services. While providing support for profiles with SQL-like 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 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.

9.0 Future work

P2P search mechanisms employ strategies different from those discussed above. NaradaBrokering's support for P2P interactions can be found in [7]. Combining P2P search mechanisms initiated by peers on the edge of the network with the schemes outlined in earlier sections, could manage the transient nature of dynamic services rather well. Research in this area and resource management would be of considerable interest. Managing interactions between Web/Grid services generated dynamically when complex tasks are initiated is another area of research.

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.

10.0 Conclusion

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.

References

1. The X Internet TechStrategy Report, May 2001 by Carl D. Howe with George F. Colony, Bill Doyle, Christopher Voce, Rebecca Shuman.
2. Zhang, L. et al. "ReSource ReserVation Protocol (RSVP) – Functional Specification", Internet Draft, March 1994.
3. Topolcic, C., "Experimental Internet Stream Protocol: Version 2 (ST-II)", Internet RFC 1190, October 1990.
4. The NaradaBrokering System http://www.naradabrokering.org
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. Grid Services for Earthquake Science. Fox et al. Concurrency & Computation: Practice & Experience.14(6-7):371-393.
10. "Integration of NaradaBrokering and Audio/Video Conferencing as a Web Service". Hasan Bulut, Geoffrey Fox, Shrideep Pallickara, Ahmet Uyar and Wenjun Wu. Proceedings of the IASTED International Conference on Communications, Internet, and Information Technology, November, 2002, in St.Thomas, US Virgin Islands.
11. "An Approach to High Performance Distributed Web Brokering". Fox and Pallickara, ACM Ubiquity 2:38. Nov 2001.
12. 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.
13. Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. Edited by Andy Oram. O'Rielly Press, CA. March 2001.
14. Bill Segall and David Arnold. Elvin has left the building: A publish/subscribe noti.cation service with quenching. In Proceedings AUUG97, pages 243–255, Canberra, Australia, September 1997.
15. Bill Segall, David Arnold, Julian Boot, Michael Henderson, and Ted Phelps. Content based routing with elvin4. In Proceedings AUUG2K, Canberra, Australia, June 2000.
16. 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.
17. Gurudutt Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Rob Strom, and Daniel Sturman. An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems. In Proceedings of the IEEE International Conference on Distributed Computing Systems, Austin, Texas, May 1999.
18. Lukasz Opyrchal et. al. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. Middleware 2000: 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. T.H. Harrison, D.L. Levine and D.C. Schmidt. The design and performance of a real-time CORBA object event service. Proceedings of the OOPSLA'97. Atlanta, GA.
22. Java Message Service Specification". Mark Happner, Rich Burridge and Rahul Sharma. Sun Microsystems. 2000. 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. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. Proceedings of Middleware 2001.
26. Squirrel: A decentralized peer-to-peer web cache. ACM PODC 2002.
27. "Collective Dynamics of Small-World Networks". D.J. Watts and S.H. Strogatz. *Nature*. 393:440. 1998.
28. "Diameter of the World Wide Web". R. Albert, H. Jeong and A. Barabasi. *Nature* 401:130. 1999.
29. A Security Framework for Distributed Brokering Systems. Shrideep Pallickara, Marlon Pierce, Geoffrey Fox, Yan Yan, Yi Huang. Available from Project URL.
30. 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.
31. XML Path Language (XPath). Version 1.0. W3C Recommendation. Available from http://www.w3.org/TR/xpath .
32. "Assertions and Protocol for the OASIS Security Assertion Markup Language," P. Hallam-Baker and E. Maler, eds. Available from http://www.oasis-open.org/ committees/security/docs/ cs-sstc-core-01.pdf.