A Preliminary Performance Evaluation of jGMA With the NaradaBrokering Framework

Distributed Systems Group, University of Portsmouth
Mark Baker, Matthew Grove, Rahim Lakhoo
[mark.baker@computer.org, {matthew.grove, Rahim.lakhoo}@port.ac.uk]

Abstract

jGMA is an extendable Java-based wide-area messaging framework designed to be a reference implementation of the GGF's Grid Monitoring Architecture. In this paper we describe jGMA's architecture and then compare its end-to-end performance against that of the NaradaBrokering framework, which is being developed by the Community Grids Lab at Indiana University. We explain our benchmarking methodology and then present the analysis of the results from these initial tests. We conclude by making some observations about the performance of jGMA and NaradaBrokering. Finally we outline our future work with jGMA to both improve its performance and enhance its functionality.

1 Introduction

Emerging Grid applications need to be provided with a range of services that fulfil the needs of their users. These services in turn rely on underlying libraries, tools, and other utilities. One important service, needed by all wide-area distributed systems, including the Grid, is mechanisms for monitoring both hardware resources and software services. These monitoring mechanisms have several roles including, gathering data (for example from instrumented applications and hardware monitors), publishing it, as well as discovering and distributing this information.

The resource monitoring framework, known as GridRM [1], requires a means of discovering and transferring data between its end-points. Existing messaging systems such as R-GMA [2] and pyGMA [3] did not meet the requirements of GridRM; this led to the development of a general purpose wide-area messaging system called jGMA, which conforms to the GGF's Grid Monitoring Architecture [4]. While GridRM motivated the initial implementation of jGMA, it has since been developed to be a flexible and extensible messaging framework.

In this paper we describe jGMA's architecture and then compare its end-to-end performance against that of the NaradaBrokering framework [5], which is being developed by the Community Grids Lab at Indiana University. In Section 2 we introduce the jGMA architecture. Section 3 is used to justify the suitability of NaradaBrokering as a system with which to compare jGMA; also we describe the benchmarks used to measure end-to-end performance. In Section 4 we explain our benchmark methodology, and present the results in Section 5. Finally we conclude the paper and describe our future work in Section 6.

2 jGMA

jGMA is a Java-based wide-area messaging framework designed to be a reference implementation of the GGF's Grid Monitoring Architecture; which is based on a publish and subscribe paradigm. jGMA consists of three core entities (see Figure 1):

1. Mediators that allow Producers and Consumers to discover each other and establish remote communications,
2. Consumers,
3. Producers.

In jGMA, Producers or Consumers can publish their existence in a directory service (registry). In turn, Producers and Consumers can use the registry to locate parties, which will act as a source or destination for the data they are interested in. Currently jGMA uses TCP Sockets for LAN communications and HTTP over the WAN. The Mediator allows wide-area connectivity for nodes that do not have direct access to the Internet; it acts effectively as a gateway into a localised jGMA installation.

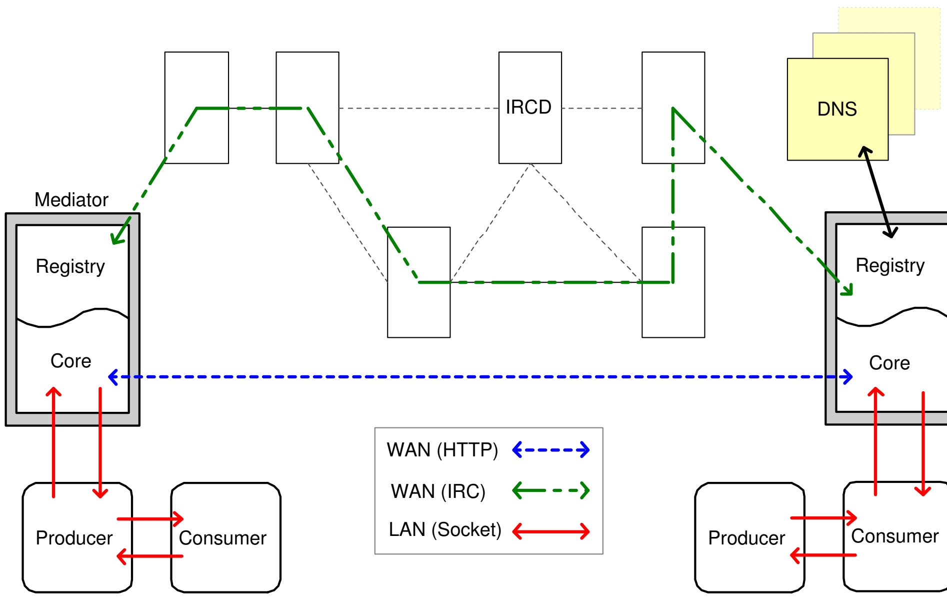

Figure 1: The jGMA Architecture.

The Virtual Registry (VR) provides Mediators with a distributed mechanism for locating other jGMA resources and routing registry queries between Mediators. The VR component provides discovery, naming, and querying services for jGMA clients. Two pluggable layers are used to make the VR extendable. The first layer allows the use of different local data stores, jGMA currently supports relational databases (via JDBC) or internal Java-based data structures. The second layer supports different communication protocols, for example TCP, HTTP, or an application specific protocol, to be used between components in the VR. The current implementation of the second layer supports the use of Internet Relay Chat (IRC) to provide service discovery and remote query routing [6]. The pluggable architecture of the VR simplifies the process of exploring how best to leverage existing technologies to create a scalable and robust VR.

3 Preliminary Benchmarking

The aim of these preliminary benchmarks is to establish the latency and bandwidth of jGMA compared to another popular system. After our initial IRC based implementation of the registry service is complete its performance and functionality will be measured and compared to other similar systems, such as R-GMA and Globus MDS [7]. Future tests will also examine the scalability of Producer and Consumer communications by investigating group communication. The popular ping-pong test was selected to measure the latency and bandwidth of sending varying message sizes with different Producers and Consumers configurations.

3.1 Selecting Software For Comparison

In order to make a good comparison we wanted to select a popular existing framework, which closely matches the feature set of jGMA. It must be written in Java, support byte messages using TCP/IP Sockets for local-area communication and provide a service to act as a router for messages over the wide-area using HTTP.

We chose NaradaBrokering, a widely used system, which is also utilised in projects such as the UK e-Science Programme [8], GlobalMMCS [9] and Anabas [10]. NaradaBrokering currently has the features that match most closely with jGMA, so its comparison is more appropriate than other technologies such as R-GMA, pyGMA, MDS and MAGGIES [11]. In the next section we introduce the NaradaBrokering framework.

3.2 The NaradaBrokering Project

The NaradaBrokering framework is a distributed messaging infrastructure, developed by the Community Grids Lab at Indiana University. Although originally designed to provide software multi-casting for real-time collaboration it now aims to provide a unified messaging environment that incorporates the capability to support Grid and Web Services, Peer-to-Peer and video conferencing, within a publish/subscribe architecture [12].

NaradaBrokering is Sun JMS compliant (version 1.02.b) [13] [14], this messaging standard allows application components to exchange unified messages in an asynchronous system. The JMS specification is used to develop Message Orientated Middleware (MOM) and defines how messages are to be communicated via queues or topics. NaradaBrokering also provides a variety of transport protocols including HTTP, TCP, NIO/TCP, UDP, and SSL.

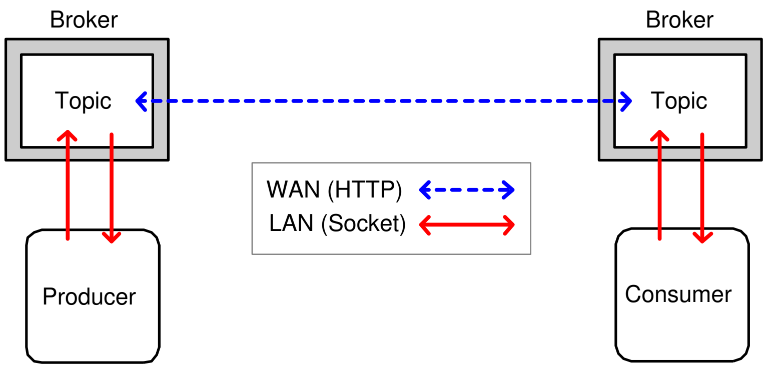

Figure 2: A Simplified View of the NaradaBrokering Architecture.

NaradaBrokering is an asynchronous messaging infrastructure with a publish and subscribe -based architecture. Networks of collaborating brokers are arranged in a cluster topology, with a hierarchy of clusters, super-clusters, and super-super-clusters [15]. Each broker is assigned a logical address within the network, which corresponds to its location and contains a Broker Node Map (BNM) for the calculation of routes, based on broker hops. The NaradaBrokering transport framework provides the capability for each link between brokers to implement a different underlying protocol [16]. The security framework incorporates an encryption key management structure, supporting a variety of algorithms, for topics, publishers, and subscribers [17]. A built-in performance aggregation service can monitor links originating from a broker and typically displays values for the average delay, latency, jitter, throughput, and loss rates [18]. Audio-video conferencing is accomplished with the aid of the Real-Time Protocol (RTP) [19] and the Java Media Framework [20] [21]. Support for JXTA Peer-to-Peer [22] end-points communicating over a NaradaBrokering broker network is propagated though a proxy [23]. NaradaBrokering also incorporates services for the compression/decompression and fragmentation/coalescing of payloads/files; it also has the ability to bypass firewalls and proxies.

Within NaradaBrokering, topics can be defined as string based, coupled with SQL like queries, `tag=value` pairs, integer based, or XML based with XPath queries. Subscriptions are organised in a hierarchical fashion, whereby brokers store client subscriptions, and broker subscriptions are recorded by super-cluster nodes, also known as cluster controllers. Subscribers implement a MessageListener, which is invoked by the broker upon receipt of a message, providing asynchronous delivery of messages. The body of a message may match one of five types, Stream, Map, Text, Object, or Bytes message. Events in NaradaBrokering are defined as time stamped messages, the broker responds to these events and routes them to the appropriate endpoint [24].

4 Benchmarking

4.1 Ping-Pong

A traditional ping-pong test was implemented in Java. This test measures the round trip time to send and receive messages of varying sizes, and can be used to assess end-to-end latency and bandwidth.

The benchmarks are written so that the time to bootstrap the test does not interfere with the measurements they gather. For NaradaBrokering we used a string based topic rather than numeric one in order to more closely match jGMA, which uses strings to identify end-points.

4.2 Topologies

Two different arrangements of Producers, Consumers and Mediators/Brokers are used to measure the latency and bandwidth of the two different types of communication.

- Test 1 measures the performance of TCP-based communications over Fast Ethernet (LAN),
- Test 2 measures the performance of simulated wide-area communications using a combination of TCP (LAN) and HTTP (WAN).

For each test the layout of NaradaBrokering and jGMA components where matched as closely as possible in order to provide comparable results (see Figure 3). The main difference is that NaradaBrokering does not allow for direct communication between end-points; data (or messages) must flow from clients via a broker(s), which handles the routing of messages. Conversely, clients in jGMA by-pass the Mediator and undertake direct communications over the host or LAN. In the first test where jGMA end-points could communicate directly the placement of the NaradaBrokering components were selected for optimal performance.

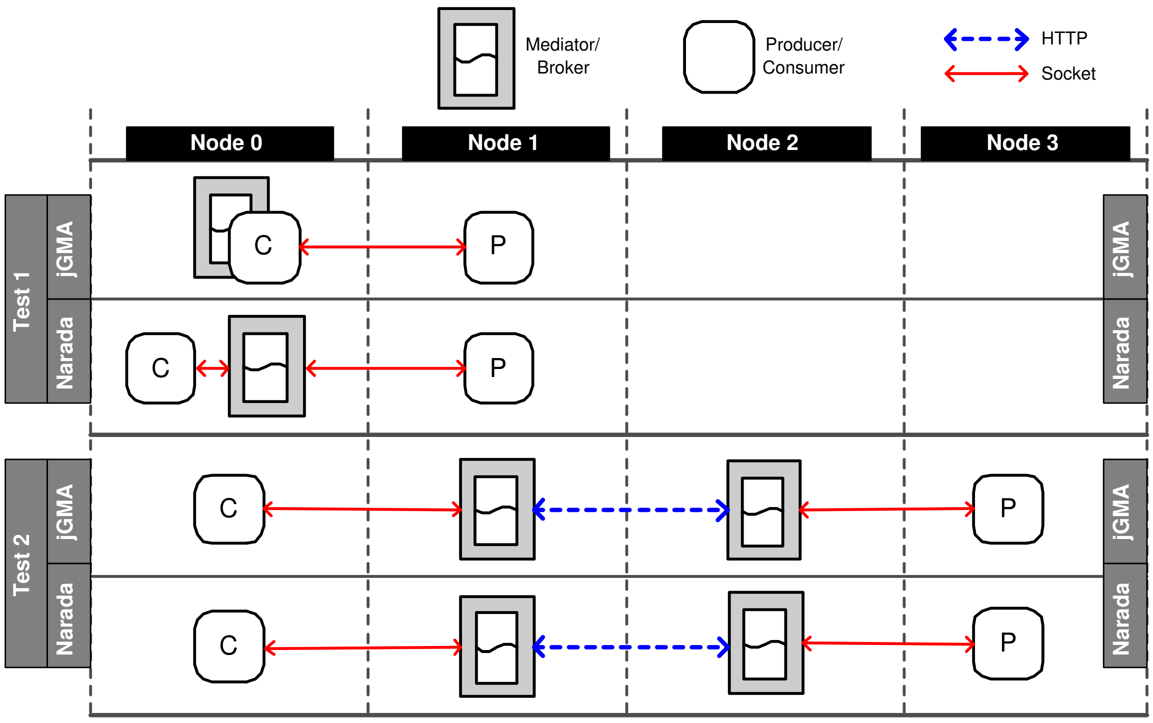

Figure 3: The Two Test Configurations.

5 System Configuration

The tests were run using four nodes from a cluster. Each node had dual 2.8GHz Xeon processors with 2 GB RAM and a 80GB EIDE disk running Debian Linux 3.1 with the 2.4.30 kernel. The machines were connected by Fast Ethernet.

For both jGMA and NaradaBrokering, the Sun JVM (Version 1.5.0-b64) was invoked with no command line options; the maximum heap size was not altered from the default (for Linux the default is 64 Mbytes). The Java timer used in the tests had microsecond resolution.

5.1 Java Baseline Performance

A Java Sockets implementation of a traditional ping-pong was used to measure the baseline communication performance. By comparing the measurements taken when timing jGMA and NaradaBrokering to the baseline performance it was possible to analyse the overhead of sending and receiving messages with each system.

6 Discussion of Results

6.1 Test 1 - LAN (Sockets)

In Test 1 the Consumer is run on one host and the Producer on another, they communicate over Fast Ethernet via Sockets. The jGMA Consumer and Producer communicate directly using Sockets only using the Mediator to bootstrap the test. The NaradaBrokering Consumer and Producer also use Sockets to send the messages, but NaradaBrokering routes messages via the broker.

6.2 Test 2 - WAN (HTTP)

In Test 2 four hosts are used to simulate a WAN. Both jGMA and NaradaBrokering components are arranged the same way, with the Consumer and Producer communicating with two separate Brokers/Mediators using Sockets and the Brokers/Mediators communicating using HTTP.

6.3 Results and Observations

The results from both tests and the baseline performance are plotted in Figures 4 and 5. Figure 4 shows the average latency with increasing message size and Figure 5 shows the bandwidth, plotted against increasing message size.

6.3.1 Latency Results And Analysis

LAN Latency:

Baseline Java has less than 200 μs latency for messages $<$512 Kbytes. Between 512 Kbytes and 4 Mbytes latency for the Java baseline test increases in step with message size. For messages $<$128 Kbytes, both jGMA and NaradaBrokering show an additional fixed latency of around 1200 μs over the baseline performance of Java. The difference between NaradaBrokering and jGMA LAN latency is negligible until 4 Kbytes, when for messages up 4 Mbytes the difference between jGMA and NaradaBrokering increases in step with message size.

Simulated WAN Latency:

For messages $<$16 Kbytes, there is a fixed 5000 μs difference in latency between jGMA and NaradaBrokering using HTTP. As the message size increases beyond 16 Kbytes the difference between the latency of the two systems decreases to 3000 μs at 128 Kbytes. From 128 Kbytes until 4Mbytes the difference in latency increases in step with message size.

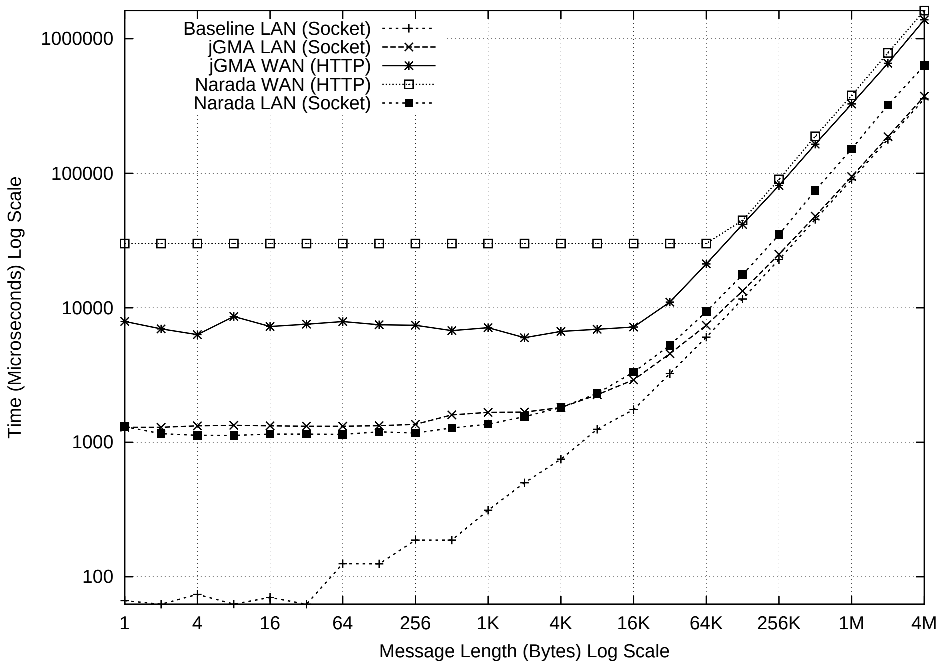

Figure 4: Latency versus Message Length.

6.3.2 Bandwidth Results And Analysis

LAN Bandwidth:

Baseline Java bandwidth utilisation peaks at 11.5 Mbytes/s. The bandwidth achieved by NaraBrokering and jGMA, for messages <8 Kbytes, is approximately 55% of that of baseline Java. After this point jGMA's bandwidth continues to increase and plateaus at 512 Kbytes with a 10.9 Mbytes/s (95.5% of the baseline bandwidth); where NaradaBrokering peaks at 128 Kbytes with a bandwidth of 7.3 Mbytes/s (65.3% of baseline), thereafter the bandwidth achieved by NaradaBrokering decreases as message size increases, falling to 56

WAN Bandwidth:

The maximum bandwidth for jGMA using HTTP communications for messages $<$ 1 Kbyte was 2.7% of the maximum achievable (0.07 Mbytes/s); NaradaBrokering achieved 0.6% (0.01 Mbytes/s). The maximum bandwidth achieved by jGMA using HTTP was 27% of the maximum (3.1 Mbytes/s); this was achieved at the 2 Mbyte point and was sustained for the 4 Mbyte message point. After peaking at 26% of the achievable bandwidth (2.8 Mbytes/s) using 128 Kbyte messages, the bandwidth utilisation for NaradaBrokering falls by 3% at the 4 Mbyte message point.

6.4 Discussion Of Results

The small difference (approximately 200 μs) between the LAN results of jGMA and NaradaBrokering for small messages is caused by the cost of jGMA opening a new Socket for every message, whereas NaradaBrokering reuses a single Socket for each message. For LAN the 1000 μs latency over that of the baseline Java is due to the internal latency of jGMA and NaradaBrokering caused by data marshalling and the cost of storing and retrieving messages from the send and receive buffers.

It is thought that the smaller fraction of the total bandwidth achieved when sending small messages is due to both frameworks using a single send thread to send messages. This means that when the available bandwidth is high, such as is the case with Fast Ethernet, and the message size is small, the maximum number of messages that can be sent per second is bound by the speed and capability of the host's CPU.

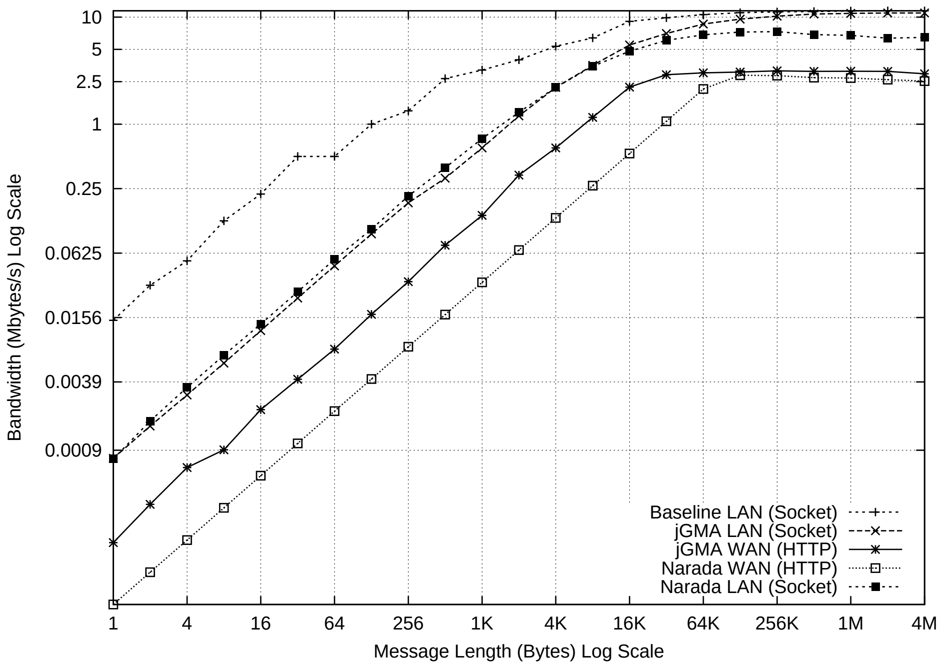

Figure 5: Bandwidth versus Message Length.

The difference between the LAN and WAN results is explained by the effect of undertaking additional, internal messaging between clients and the mediators/brokers, and the overheads of processing the HTTP encapsulated message. The overhead is greatest for messages of <1 Kbyte.

7 Summary And Conclusions

In this paper we have briefly described jGMA and NaradaBrokering, and also presented the results from an initial set of benchmarks establishing point-to-point performance comparing the two systems. These benchmarks have provided some insight into the expected performance and capabilities of jGMA.

The end-to-end performance tests show, that on a LAN for messages less than 2 Kbytes, jGMA and NaradaBrokering have comparable performance. jGMA achieves 95% of the maximum bandwidth utilisation for larger messages. Reusing Sockets rather than creating a new one for each message can further reduce the latency of jGMA

When using HTTP with messages <128 Kbytes, there is a constant latency difference between jGMA and NaradaBrokering of between 3000-5000 μs. For larger messages this difference increases. The poor HTTP performance relative to that of Sockets provides scope for further improvement of both the jGMA and NaradaBrokering.

7.1 Future Work

The results of these simple benchmarks have shown a number of deficiencies in the internal programming of jGMA; we intend to improve the performance of the implementation both the Socket and HTTP programming. We will then move on to system test the scalability jGMA components by varying the number of Producers and Consumers, and determining how many messages per second a client can handle. Varying the number of Producers sending messages to a Consumer will help assess the scalability of jGMA by simulating a Consumer under heavy load.

The current focus of jGMA research is the distributed registry service. We are currently developing a pluggable registry component (the Virtual Registry), which will allow us to explore how best to leverage existing P2P technologies to create a scalable, robust registry service to provide mechanisms that allow jGMA components to locate and query each, over the wide-area. The Virtual Registry will be benchmarked and compared to other service discovery systems such as R-GMA and Globus MDS.

References

[1] GridRM, http://gridrm.org/

[2] R-GMA, http://www.r-gma.org/

[3] pyGMA, http://www-didc.lbl.gov/pyGMA/

[4] GMA, http://www.ggf.org/documents/GFD/GFD-I.7.pdf

[5] Community Grids Lab, Indiana University, NaradaBrokering, http://www.naradabrokering.org

[6] M.A. Baker and M. Grove, A Scalable Registry Service For jGMA, Work-in-Progress Novel Grid Technolgies, CCGrid 2005, March 2005.

[7] Globus MDS, http://www-unix.globus.org/toolkit/mds/

[8] National e-Science Centre, http://www.nesc.ac.uk/

[9] GlobalMMCS, http://www.globalmmcs.org/

[10] Anabas, Inc. eLearning and collaboration, http://www.anabas.com/netscape/index.html

[11] MAGGIS, http://www.cs.uiowa.edu/ apadmana/MAGGIS/

[12] Fox. G, Pallickara. S, and Parastatidis, S. Towards Flexible Messaging for SOAP Based Services, Proceedings of the IEEE/ACM Supercomputing Conference 2004, Pittsburgh, PA.

[13] Sun Microsystems, Java, Java Messaging Service (JMS) Specification 1.0.2b, http://java.sun.com/products/jms/

[14] Fox. G, and Pallickara. S. JMS Compliance in the Narada Event Brokering System, Proceedings of the 2002 International Conference on Internet Computing (IC-02), Volume 2 pp 391-397.

[15] Pallickara. S, and Fox. G., On the Matching Of Events in Distributed Brokering Systems, Proceedings of IEEE ITCC Conference on Information Technology, April 2004. Volume II pp 68-76.

[16] Pallickara. S, Fox. G, Gunduz. G. Y, Liu. H, Uyar. A, and Varank. M., A Transport Framework for Distributed Brokering Systems, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, (PDPTA'03),Volume II pp 772-778.

[17] Pallickara. S, Pierce. M, Yan. Y, and Huang. Y., A Security Framework for Distributed Brokering Systems. Technical Report, http://www.naradabrokering.org/papers/NB-SecurityFramework.pdf

[18] Gunduz. G, Pallickara. S, and Fox. G., A Portal Based Approach to Viewing Aggregated Network Performance Data in Distributed Brokering Systems, Proceedings of the 2003 International Conference on Internet Computing, Volume II pp 495-501.

[19] RTP: A Transport Protocol for Real-Time Applications (IETF RFC 1889), http://www.ietf.org/rfc/rfc1889.txt

[20] Sun Microsystems, Java, Java Media Framework (JMF), http://java.sun.com/products/java-media/jmf/

[21] Uyar. A, Pallickara. S, and Fox. G., Towards an Architecture for Audio Video Conferencing in Distributed Brokering Systems, Proceedings of the 2003 International Conference on Communications in Computing pp 17-23.

[22] JXTA, http://www.jxta.org/

[23] Fox. G, Pallickara. S, and Rao. X.. Scaleable Event Infrastructure for Peer to Peer Grids, Proceedings of the ACM Java Grande ISCOPE Conference 2002.pp 66-75. Seattle, WA.

[24] Pallickara. S, and Fox. G., A Scheme for Reliable Delivery of Events in Distributed Middleware Systems, Proceedings of the IEEE International Conference on Autonomic Computing, New York, NY. pp 328-329.