A Study of Publish/Subscribe Systems for Real-Time Grid Monitoring

Chenxi Huang, Peter R. Hobson, Gareth A. Taylor, Paul Kyberd
School of Engineering & Design, Brunel University, Uxbridge, UK
chenxi.huang@brunel.ac.uk

Abstract - **Monitoring and controlling a large number of geographically distributed scientific instruments is a challenging task. Some operations on these instruments require real-time (or quasi real-time) response which make it even more difficult. In this paper, we describe the requirements of distributed monitoring for a possible future Electrical Power Grid based on real-time extensions to Grid computing. We examine several standards and publish/subscribe middleware candidates, some of which were specially designed and developed for Grid monitoring. We analyze their architecture and functionality, and discuss the advantages and disadvantages. We report on a series of tests to measure their real-time performance and scalability.**

Index terms – **monitoring, real time systems, distributed computing, Grid Computing, publish/subscribe systems**

I. INTRODUCTION

In the foreseeable future, there will be a larger number of small power generators that use renewable energy sources. They will be highly dispersed in different physical locations. Monitoring and control of the ordinary activities of such generators will become an important issue [1]. Power generators typically produce data for monitoring that can be collected remotely and updated frequently. Up to tens of thousands of power generators will be distributed all over the UK, some of them will only be connected through a low speed ADSL line. Data monitoring must be processed in real-time in order to accurately coordinate and control the generators. For example, if a power generator has been switched on but does not respond for a long time then it will be considered to be malfunctioning. A real-time system does not need to be very fast but should be stable and respond within a reasonable predefined time limit. Power Grid monitoring is a distributed soft real-time monitoring system. Most of the data for monitoring should be received within a time limit (e.g. 5 seconds). A small number of delays are sometimes allowed (e.g. less than 0.5%). Traditional monitoring systems are highly centralized and run on dedicated Wide Area Networks (WANs). Considering the distributed nature and the large number of the generators, this solution is expensive and will not scale very well.

The GRIDCC (Grid enabled Remote Instrumentation with Distributed Control and Computation) project [2] aims to realize distributed monitoring and control via Grid Computing. Other use cases include CMS data acquisition system and Synchrotron Radiation Storage Ring Elettra [3]. One of the objectives is to find a scalable distributed monitoring solution that could satisfy the soft real-time requirements.

The paper is organized as follows: We examine and analyze the functionality and architecture of publish/subscribe middleware candidates in section II, describe the experiments and analyze the results in section III, review related work in section IV and conclude in section V.

II. PUBLISH/SUBSCRIBE SYSTEMS

A publish/subscribe (pub/sub) system is a many-to-many data dissemination system. Publishers publish data and subscribers receive data that they are interested in. Publishers and subscribers are independent and need to know nothing about each other. The middleware delivers data to its destination. The middleware's functionality is more than forwarding data from source to destination. It provides advanced functions like data discovery, dissemination, filtering, persistence and reliability, etc. Data are discovered through the middleware and can be transferred either directly from publisher to subscriber or via a broker. The subscriber can be automatically notified when new data becomes available. Compared to a traditional centralized client/server communication model, pub/sub system is asynchronous and is usually distributed and scalable.

Considering the distributed nature of the information provider and consumer, and the high performance requirement of our problem, pub/sub systems seem to be the best solution towards distributed monitoring. Several proposals and implementations have been developed for Grid monitoring. We examine these candidates in the following sections.

A. GMA and R-GMA

Recognizing the complicated nature of a monitoring system for Grid Computing, the Global Grid Forum

(GGF) proposed the Grid Monitoring Architecture (GMA) [4] as a solution. The objective of GMA is to facilitate the development of interoperable and high performance monitoring middleware.

GMA divides a pub/sub middleware into three basic components: producer, consumer and directory service. A producer gathers data from various sources, such as an instrument or computer server. A consumer receives data from a producer and forwards them to destination. By separating data discovery from data transfer, GMA ensures scalability and performance. Data discovery is through a directory service. The directory service is an information service where a producer or consumer publishes its existence and relevant metadata to. Consumer may search directory for the producer that it is interested in. Then they can establish a connection and transfer data directly. GMA proposes three data transfer modes between producer and consumer: publish/subscribe, query/response, and notification. In the publish/subscribe mode, either a producer or consumer can initiate data transfer. The producer sends data continuously and either side can terminate. In the query/response mode, a consumer initiates communication and the producer sends all the data to the consumer in one response. In the notification mode, the producer must be the initiator. The producer sends all the data to the consumer in one notification.

The Relational Grid Monitoring Architecture (R-GMA) [5] is an implementation of GMA. The novel design of R-GMA is that it has a large virtual database (Fig. 2) which looks and operates like a conventional relational database. It supports a subset of the standard SQL language. Data are published using SQL INSERT statement and queried using SQL SELECT statement. The difference between a virtual database and conventional relational database is that a virtual database has no central storage and data are distributed all over the network.

Data discovery is through registry and schema. Producers and consumers register their addresses in the registry. Data must be disseminated via the producer and consumer to reach destination. Data transfer between consumer and destination is query/response only.

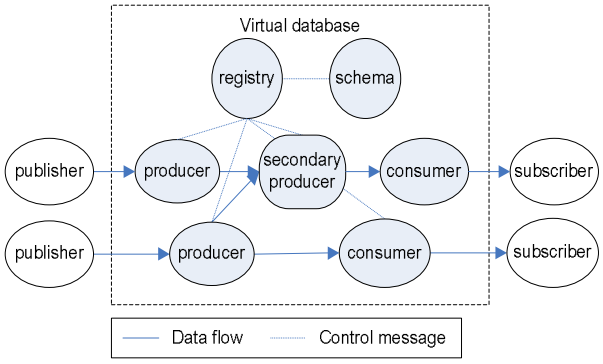

Fig. 2. R-GMA virtual database

R-GMA conforms to Web Services Architecture. It uses SOAP messaging over HTTP/HTTPS and Java Servlet technology to exchange request/response (except data streaming which is implemented in a more efficient way). R-GMA APIs are available in Java, C, C++ and Python.

B. *JMS and NaradaBrokering*

Java Message Service (JMS) [6] is a widely accepted industry standard that aims to simplify the effort needed for applications to use Message Oriented Middleware (MOM). JMS defines a set of Java APIs (Application Programming Interfaces), with which Java programmers can send and receive messages via MOM in a uniform and vendor-neutral way regardless of what the actual underlying middleware is.

Data are discovered by *destination*. There are two kinds of *destinations*: *queue* and *topic*. Data are wrapped in a JMS message. JMS supports two data dissemination modes: Point-To-Point (PTP) (brokerless) and publish/subscribe (brokered). Messages are delivered via a topic. JMS supports synchronous and asynchronous data transfers. For synchronous transfer, the subscriber can either poll or wait for the next message. For asynchronous delivery, the subscriber registers itself as a listening object, and the publisher will automatically send message by invoking a method of the subscriber (callback).

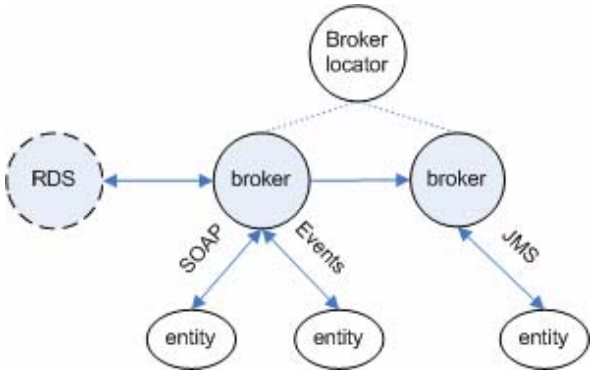

Fig. 1. NaradaBrokering network map

NaradaBrokering [7] is an open source, distributed messaging infrastructure. It is fully compliant with JMS. NaradaBrokering supports SOAP message, JMS message and complicated events. NaradaBrokering supports PTP and pub/sub data dissemination modes, and synchronous and asynchronous data transfer modes proposed by JMS. NaradaBrokering supports a

number of Web Services and Grid Services specifications, such as WS-Resource Framework (WSRF), WS-Notification and WS-Eventing.

Several brokers can form a Broker Network Map (BNM) (Fig. 1). A specialized node called Broker Discovery Node (BDN) can discover new brokers. NaradaBrokering has a very efficient algorithm to find a shortest route to send the events to the destination in a BNM. NaradaBrokering is a very fast message dissemination middleware and it has been successfully adopted for audio/video conferencing. NaradaBrokering supports a number of underlying data transport protocols, including blocking and non-blocking TCP, UDP, multicast, SSL, HTTP, HTTPS and Parallel TCP streams.

III. EXPERIMENTS

We chose NaradaBrokering and R-GMA as candidates because NaradaBrokering is an open-source, high performance middleware. It is JMS compliant and has been successfully used for video/audio conferencing. R-GMA has been developed for Grid monitoring. It has very good scalability and provides useful functionality like latest and historical query, content based filtering, etc. We developed Power Grid simulation programs. We also measured the real-time performance, throughput and scalability of the candidates.

A. Test environment

The Hydra cluster consists of 8 identical computer nodes, (Hydra1 to Hydra8). They are interconnected with each other using a 100Mbps switch to setup a private LAN. The LAN is isolated from outside to ensure the validity of the test results. The actual data transfer rate within the LAN is 7 ~ 8 Mbytes per second (tested and reported by Linux sftp). Our tests were all performed in Hydra cluster. We installed Linux, Java and testing software on Hydra nodes. The hardware specifications and software versions are listed in Table I. NaradaBrokering is written in Java and requires Java Virtual Machine. R-GMA is implemented as a Java Servlet and requires Tomcat, MySQL and the Java Virtual Machine.

B. Power Grid simulation

We have developed a Java program to simulate the activities of a large number of distributed power generators. It could fork into a large number of threads. Each thread may simulate one power generator and generate monitoring data, such as power output and voltage. These monitoring data were published to the middleware periodically at a specified frequency (e.g. every 10 seconds). Another Java program received data from the middleware. Information of the monitoring data (such as sending and receiving time, etc) was dumped into a local text file for later analysis. Data dumping used highly efficient logging APIs which were mainly cached hard-drive write operations, so the overhead was negligible. When simulating 750 generators on one computer, the publishing rate was 75 messages per second and the throughput was less than 50Kbytes per second. The CPU idle time was above 85%. For most tests, we simulated no more than 750 generators on one computer. We simulated 1000 generators per computer in only one test and the result seemed to be consistent.

TABLE I HARDWARE SPECIFICATIONS AND SOFTWARE VERSIONS

CPU and memory	OS and JVM	Middleware
PentiumIII 866MHz, 2GB	Sci Linux, kernel 2.4.21, Sun Hotspot JVM 1.4.2	NaradaBrokering v1.1.3, RGMA gLite v3.0, Tomcat v5.0.28

C. Performance metrics

We used the following parameters to measure performance [8] [10]: Round-Trip Time (RTT), RTT variation, loss rate and percentile of RTT. RTT was calculated as the mean round-trip time of all the messages. The round-trip time of each message was the difference between sending and receiving time. RTT variation was calculated as the standard deviation (STDDEV) of all the round-trip times. Percentile of RTT was the percentage of the round-trip times.

Publishing and subscribing are asynchronous operations, which consist of two synchronous operations. Response time is the time to complete one synchronous operation, which is the time it takes to send or receive a message.

We recorded CPU idle time and memory consumption using Linux tool *vmstat*. CPU idle time was calculated as the average of CPU idle time during the tests. Memory consumption was calculated as the difference between peak and bottom values. It should be noted that memory consumption was sometimes not very accurate because Linux used some memory as cache.

D. Why not Web Services

We did not use Web Services to test the candidate middlewares mainly for the reason of performance. Web Services are known to be slow and not suitable for high performance scientific computing [9]. The serialization and de-serialization of XML and floating point value/ASCII conversion are the bottlenecks. The interoperability issue can be compensated by introducing a *proxy* that has a Web Services interface [3].

E. NaradaBrokering tests

Simulated power generators were created at the interval of 0.5 second. Each generator first slept for a random time between 10 to 20 seconds to allow the monitoring data to distribute evenly, it then used a JMS TopicPublisher to publish data to a JMS topic at the interval of 10 seconds. Two integer, five float, two long, three double and four string values were packaged in a JMS MapMessage as monitoring data. Another Java program used JMS notification mechanism to receive monitoring data from the same topic. It created a *listener* and subscribed to the topic with a simple JMS selector (e.g. "id<10000"). This selector did not filter out any data but just to simulate real uses. The *listener* would be automatically notified by Narada broker when new messages become available. All the tests used non-persistent delivery, non-durable subscription, non-transaction, non-priority and AUTO_ACKNOWLEDGE settings unless otherwise indicated.

In order to create more than 1000 threads, we used Linux command to set file descriptors to 50000 ('ulimit -n 50000'). We allocated 1GB memory for Java Virtual Machine of NaradaBrokering ('-Xms1024m -Xmx1024m').

1) Comparison tests:

The aim of these tests was to measure how different underlying transport protocols, payload and concurrent connections could affect performance. We simulated 800 power generators. Each test lasted 30 minutes and was performed twice to ensure validity. The settings of the tests are listed in TABLE II. All tests used AUTO_ACKNOWLEDGE except test 2, which used CLIENT_ACKNOWLEDGE. Test 5 (Triple) used triple payload and the publishing rate was reduced to 1/3, therefore the total data delivered remained the same. Test 6 (80) used 80 generators (concurrent connections), which is 1/10 of the other test. The publishing rate was increased 10 times, therefore the total data delivered remained the same.

TABLE II COMPARISON TESTS SETTINGS

	Transport protocol	ACK mode	comment
Test1 (UDP)	UDP		
Test2 (UDP CLI)	UDP	CLIENT	
Test3 (NIO)	NIO		
Test4 (TCP)	TCP		
Test5 (Triple)	TCP		Triple payload
Test6 (80)	TCP		80 connections

In test 1 (UDP), a total of 144,000 messages were sent and 143,914 messages were received. The loss rate was 0.06%. In test 2 (UDP CLI), the loss rate was 0.03%. For all other tests, the loss rate was zero.

The results of the tests (fig. 3 & fig. 4) show that TCP is a very stable transport protocol and has excel-

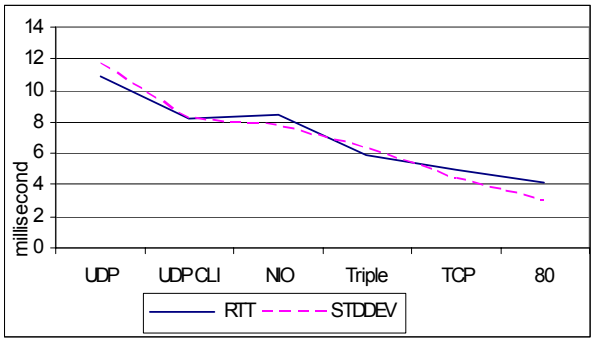

Fig. 3. Narada comparison tests
Round-Trip Time and Standard Deviation

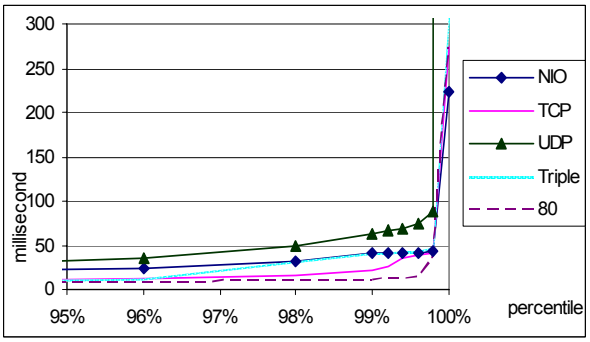

Fig. 4. Narada comparison tests, percentile of RTT

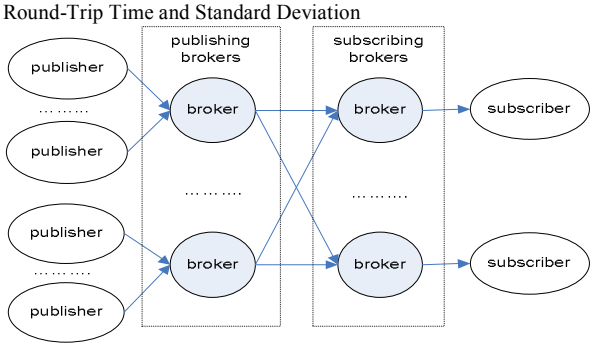

Fig. 5. Distributed architecture

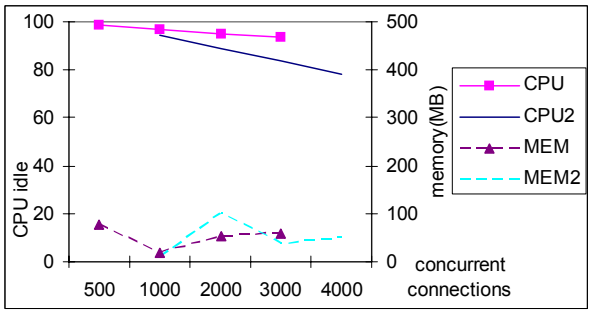

Fig. 6. Narada tests, CPU idle and memory consumption
CPU and MEM are single server tests
CPU2 and MEM2 are DBN tests

lent performance. The results of UDP test are surprisingly high. The possible reason is that we used JMS over UDP. UDP is connectionless which has no guarantee whether a packet will be received or not, but JMS requires an acknowledgement. The way that Narada acknowledges the messages severely slows the performance down. Test 5 (Triple) and test 6 (80) were aimed to compare the effect of the size of the messages and the number of concurrent connections to performance. The performance slowed down with large payload. This implies that Narada is good at small sized messages. The percentile of RTT is shown in fig. 6.

2) Performance and scalability tests:

We simulated a number of generators to test the performance of a single Narada broker and Distributed Broker Network. The test settings were the same as previous TCP test (test 4).

We setup a distributed topology which could take advantage of middleware's capability to efficiently route messages to destinations. This topology is scalable. It can sustain an even larger number of concurrent connections and maintain a good performance (fig. 5). Brokers are components of pub/sub middleware. They are located on different computers and interconnected with each other to comprise a broker network. Publishers connect to publishing brokers. Subscribers connect to subscribing brokers. Messages published to any one of the brokers can be received by any subscriber who is also connected to the network and has subscribed data. A publishing broker accepts no more than *m* concurrent connections. A subscribing broker accepts throughput of no more than *n*. If *m* and *n* are the safe thresholds for a broker and will not cause "out of memory" error or severe performance slow down, then this topology is able to maintain a large number of concurrent connections. We used four nodes to setup a Distributed Broker Network (DBN). One of them was the unit controller and assigned addresses to the other three nodes. We used the other four nodes to simulate generators and publish data. Data were received by the node where they were sent and there was no time synchronization problem.

Fig. 7, fig. 8 and fig. 9 show the test results. Narada performed very well. 99.8% of messages arrived within 100 milliseconds. There was a smooth increase of round-trip time according to the number of concurrent connections. Both round-trip time and standard deviation were very low. Our tests also showed that on our testbed a single Narada broker cannot accept 4000 concurrent connections. It ran out of memory to create new threads to serve more incoming connections. The DBN could accept more than 4000 concurrent connections and maintain a good performance.

Round-trip time and standard deviation are com-

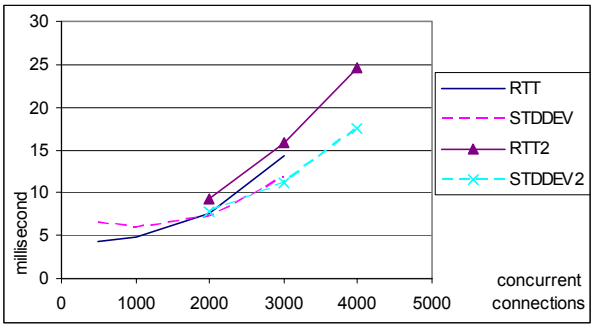

Fig. 7. Narada tests, round-trip time and standard deviation
RTT and STDDEV are standalone server tests
RTT2 and STDDEV2 are DBN tests

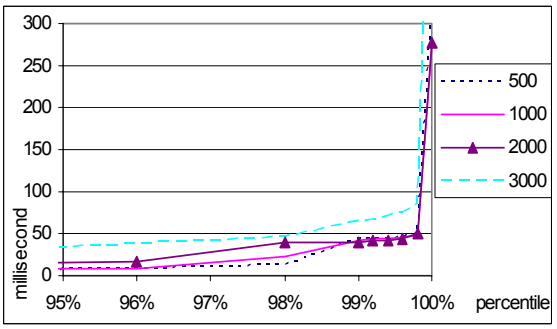

Fig. 8. Narada single server tests, percentile of RTT
For a number of 500 ~ 3000 concurrent connections

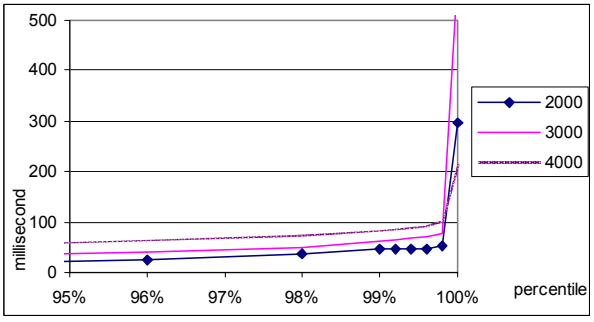

Fig. 9. Narada DBN tests, percentile of RTT
For a number of 2000 ~ 4000 concurrent connections

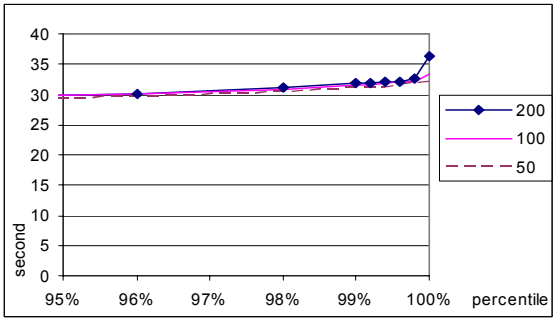

Fig. 10. R-GMA Primary and Secondary Producer tests
For a number of 50 ~ 200 concurrent connections
percentile of RTT

pared in fig. 7. We expected the results of DBN tests to be better than single server. However, they are a little disappointing and are higher than single server. We have monitored unnecessary data flow between nodes, that is, data flowed to a node even if there was no subscriber linked to it. This unnecessary data flow slowed performance down. CPU idle and memory consumption were recorded in fig. 6. CPU load of the DBN tests was higher than single server. All these strongly suggested that data were broadcast and not diverged to different routes. It would be difficult for Narada DBN to accept more connections and higher throughput.

3) Summary:

Narada has excellent real-time performance and high throughput. We recommend TCP as the underlying transport protocol to reach high performance. Both message size and publishing rate affect performance. A single Narada broker could not accept more than 2500 concurrent connections on our testbed. A Distributed Broker Network could support a larger number of concurrent connections. But the current version has some deficiency that limits its scalability.

F. R-GMA tests

Simulated power generators were created at an interval of 1 second. Each generator waited for a random time between 10 to 20 seconds to allow publishing data to distribute evenly and to allow R-GMA server enough time to 'warm up'. The generator then used Primary Producer API to publish monitoring data into a table at the interval of 10 seconds. Primary Producers used memory storage to allow fast query. The latest retention period was set to 30 seconds and history retention period was set to 1 minute. We used four integer, eight double and four char (length 20) values, which were wrapped in an SQL statement, as monitoring data. Consumer used continuous query to receive data from Primary Producers. Another Java program (subscriber) used Consumer API to receive data from the Consumer. The subscriber could not be automatically notified by the Consumer and it queried the Consumer at the interval of 100 milliseconds. Therefore there was a 100 millisecond error. Monitoring data were received by the machine where they were sent and there was no time synchronization problem.

R-GMA server ran within Tomcat. The number of concurrent connection of Tomcat was increased to 1000. Memory allocated to Java Virtual Machine was increased to 1GB ('-Xmx1024m'). R-GMA used non-secure mode and the underlying transport protocol was

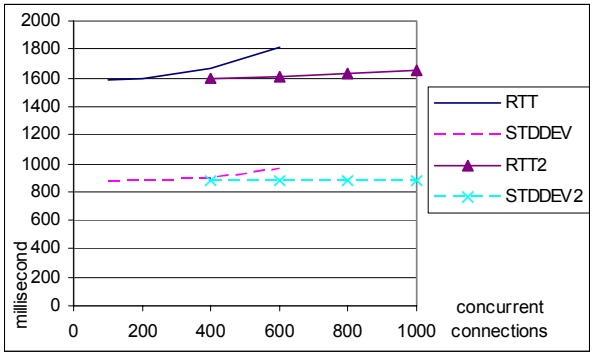

Fig. 11. R-GMA Primary Producer and Consumer tests
RTT and STDDEV are tests on single server
RTT2 and STDDEV2 are distributed network tests

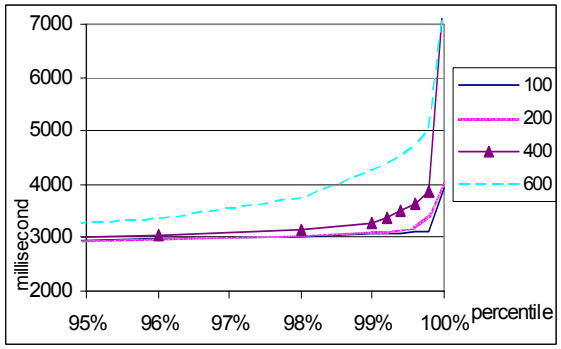

Fig. 12. R-GMA Primary Producer and Consumer single server tests
For a number of 100 ~ 600 concurrent connections
percentile of RTT

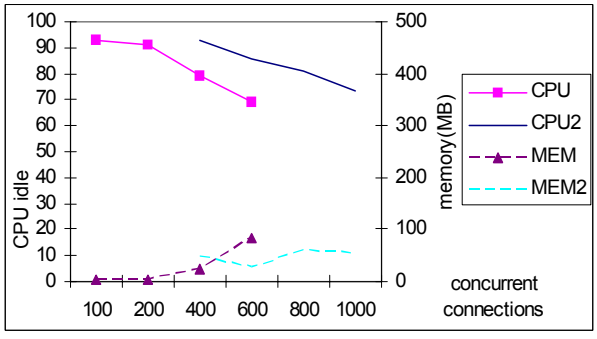

Fig. 13. RGMA Consumer tests, CPU idle and memory consumption
CPU and MEM are tests on single server
CPU2 and MEM2 are distributed network tests

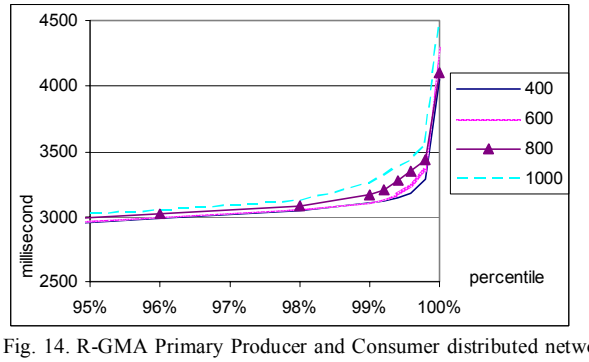

Fig. 14. R-GMA Primary Producer and Consumer distributed network tests, percentile of RTT
For a number of 400 ~ 1000 concurrent connections

HTTP.

1) Performance and scalability tests:

We simulated a number of generators to test R-GMA components on a single server and a distributed R-GMA architecture.

R-GMA has a natural way to implement a distributed architecture. The R-GMA Producer, Consumer and Registry can be installed onto different machines. The Producer gathers monitoring data and the Consumer sends data to the subscribers. R-GMA was installed onto four Hydra nodes: two Producer nodes and two Consumer nodes. R-GMA client and power generator simulation programs were installed on the other four Hydra nodes, two nodes published data and two received data. The time of the computers were synchronized by NTP (Network Time Protocol).

R-GMA supports two underlying application layer protocols for message transfer: HTTP and HTTPS. We did not use HTTPS because of the encryption overhead. In our tests we found that when creating a large number of Primary Producers, each thread must wait for a short time (5 ~ 10 seconds) before publishing data otherwise data will probably be lost. This is probably because it took some time for the producer to look for the consumer. We have tested 400 generators publishing data without waiting for the server to 'warm up'. A total of 72,000 messages were sent and 71,876 messages were received. The loss rate was 0.17%.

The results of the Primary Producer and Secondary Producer tests are shown in fig. 10. The delays were up to 35 seconds. The results of the Primary Producer and Consumer tests are shown in fig. 11, fig. 12 and fig. 14. The loss rates were zero for all tests. Both round-trip time and standard deviation were higher than those of Narada tests. 99% of messages arrived within 4000 milliseconds. Our tests also showed that one R-GMA server cannot accept 800 concurrent connections. It ran out of memory to create new threads to serve incoming connections.

Comparing the results, distributed architecture performs better than a single server. The distributed architecture can accept up to 1000 concurrent connections (and could be even more). CPU idle and memory consumption are shown in fig. 13. CPU load of a distributed architecture is lower than a single server. The results strongly suggest that R-GMA scales very well.

2) Round-trip time decomposition:

In order to further analyze the performance of R-GMA, we decompose Round-Trip Time into three phases. PRT is Publishing Response Time, which is how long it takes to publish data. PT is Process Time, which is how long it takes to process data in the middleware. SRT is Subscribing Response Time, which is how long it takes to receive data when it is available. Their relationship is represented in the following equation:

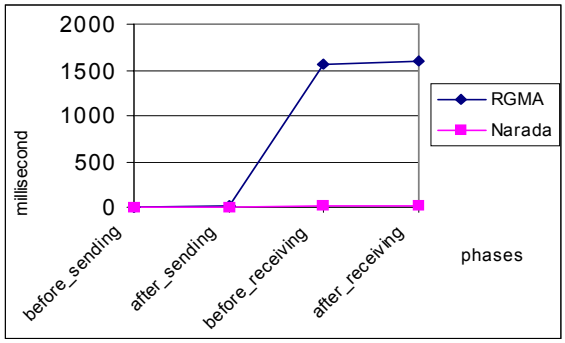

Fig. 15. RTT decomposition

$$RTT = PRT + PT + SRT$$

Fig. 15 shows these phases of R-GMA and NaradaBrokering. PRT is before_sending ~ after_sending in the graph. PT is after_sending ~ before_receiving. SRT is before_receiving ~ after_receiving. As we can see from the graph, both Publishing and Subscribing Response Time of R-GMA are short, but the Process Time is very long. This long delay occurs in the Primary Producer and Consumer. The three phases of NaradaBrokering are very short.

3) Summary:

R-GMA has lower performance and throughput than NaradaBrokering, because R-GMA takes a long time to process data. A Primary Producer should wait for a few seconds to let R-GMA server 'warm up' to avoid data delay and lost. A single R-GMA Producer could not accept more than 500 concurrent connections on our testbed. R-GMA has very good scalability. A distributed R-GMA network has better performance and can deal with a larger number of concurrent connections.

We find discrepancies between our test results and [11], where the authors achieved high performance with R-GMA. This is because we tested different versions of R-GMA. They tested an old API of R-GMA (Stream Producer and Archiver) and we tested a newer version (Primary Producer, Secondary Producer and Consumer). We contacted R-GMA developers and found that there was now a deliberate delay of 30 seconds in the Secondary Producer. However, the delays in the Primary Producer and Consumer need further investigation.

IV. Related work

IBM Reliable Multicast Messaging (RMM) [12] is a high-throughput low-latency publish/subscribe middleware. Their tests show that RMM is the fastest Message Oriented Middleware (MOM) available in the market. The performance of publish/subscribe middleware is usually restricted by the size and quantity of the messages delivered. Their study shows that in MOM the quantity of the messages is the dominant overhead. RMM achieves high performance by using message aggregation. Message aggregation is to reduce the number of total messages by combining several messages addressed to the same destination into one big message. Message aggregation can be accomplished either at the sender side or the middleware side.

X. Zhang et al. [13] have tested and compared three Grid monitoring systems, which were MDS, R-GMA and Hawkeye. They distinguished four components of the systems and tested their throughput and response time separately. Their test results show that different components of different middleware have different performance.

V. CONCLUSION AND FUTURE WORK

We explained the requirements of real-time monitoring of the future Power Grid. We examined the architecture and functionality of several standards and pub/sub middlewares. The middlewares are all decentralized and Web Service compliant. We carried out tests to measure their performance. The test results show that NaradaBrokering has very good real-time performance, high throughput, and average scalability. R-GMA has lower real-time performance, lower throughput and very good scalability (Table III). Considering additional network and application delays, the current version of R-GMA is not suitable for real-time monitoring. We are working with R-GMA developers to improve performance in this area.

We have found a deficiency in the current version of NaradaBrokering, which causes data congestion and limits its scalability. We have contacted the developers and are going to test the newest release in the future. Related work shows that an old version of R-GMA has improved performance compared to the current release. We have contacted R-GMA developers and so far found a performance bottleneck. However, further investigation needs to be conducted.

TABLE III R-GMA AND NARADABROKERING COMPARISON

	Real-time performance	Concurrent Connections & Throughput	Scalability
R-GMA	Average	Average	Very good
Narada	Very good	Very good	Average

Another important conclusion that can be drawn from this research is that when evaluating middleware for real-time monitoring applications an important consideration is the efficiency of the middleware to locate resources within a predefined time limit. If a real-time monitoring resource is not located within a predefined time limit then data loss can occur.

R-GMA has many advantages over other MOM middleware. It provides content-based filtering, latest and history query, etc. For applications where there is no such strict real-time requirement, R-GMA will be considered.

ACKNOWLEDGEMENT

This work is funded by the GRIDCC EU project under contract number 511382 and is presented on behalf of the GRIDCC consortium.

REFERENCES

[1] G. A. Taylor, M. R. Irving, P. R. Hobson, C. Huang, P. Kyberd and R. J. Taylor, "Distributed Monitoring and Control of Future Power Systems via Grid Computing", IEEE PES General Meeting 2006, Montreal, Quebec, Canada, 18-22 June 2006.

[2] Grid enabled Remote Instrumentation with Distributed Control and Computation (GRIDCC), www.gridcc.org

[3] Frizziero, E. Gulmini, M. Lelli, F. Maron, G. Oh, A. Orlando, S. Petrucci, A. Squizzato, S. Traldi, S. "Instrument Element: a new grid component that enables the control of remote instrumentation", IEEE international Symposium on Cluster Computing and the Grid, Singapore, May 2006.

[4] B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, R. Wolski, "A Grid Monitoring Architecture", http://www.gridforum.org/documents/GFD.7.pdf

[5] A.W.Cooke et al., "The Relational Grid Monitoring Architecture: Mediating Information about the Grid", Journal of Grid Computing, vol 2 no 4 December 2004

[6] Java Message Service specification, version 1.1, available from http://java.sun.com/products/jms

[7] NaradaBrokering, http://www.naradabrokering.org

[8] Lowekamp, B., Tierney, B., Cottrel, L., HUGHES- JONES, R., Kielemann, T., and Swany, M. "A hierarchy of network performance characteristics for grid applications and services", Tech. Rep. Recommendation GFDR. 023, Global Grid Forum, May 2004.

[9] K. Chiu, M. Govindaraju, and R. Bramley. Investigating the limits of SOAP performance for scientific computing. In Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, pages 246–254, 2002.

[10] C. Demichelis and P. Chimento, "IP Packet Delay Variation Metric for IP Performance Metrics (IPPM)", IETF RFC3393, Nov 2002.

[11] R. Byrom et al, "Performance of R-GMA for Monitoring Grid Jobs for CMS Data Production" 2005 IEEE Nuclear Science Symposium Conference Record pp. 860-864 (2005).

[12] B. Carmeli, G. Gershinsky, A. Harpaz, N. Naaman, H. Nelken, J. Satran and P. Vortman, "High Throughput Reliable Message Dissemination", ACM Symposium on Applied Computing, Nicosia, Cyprus, Mar 2004

[13] X. Zhang, J. Freschl, and J. Schopf, "A Performance Study of Monitoring and Information Services for Distributed Systems", Proceedings of HPDC, August 2003.