Fast Information Transport for an Instrument Enabled Grid

E. Frizziero[#], Z. Har'El[*], F. Lelli[#], B. Mandler[*], G. Maron[#], P. Molini[#], S. S. Pinter[*]

[#]*INFN - Laboratori Nazionali di Legnaro,*
Viale dell'Università 2, 35020 Legnaro (PD), ITALY
{frizziero, lelli, maron, molini}@lnl.infn.it
[*]*IBM - Haifa Research Lab., Haifa University Campus,*
Mount Carmel, Haifa, 31905, Israel
{zharel, mandler, shlomit}@il.ibm.com

Abstract

Grid-based computing frameworks leverage underutilized processing and storage resources. We present and evaluate a new high-performance, reliable middleware layer that can incorporate instruments into a grid. This Java based messaging system supports remote distributed control and operation of scientific instruments, such as sensors and probes, thereby significantly expanding the grid's capabilities.

Various comparative measurements show that our system outperforms the top-ranked publish-subscribe Java systems in the market. Our software can reach a peak message exchange rate of 900,000 messages per second, with a latency of less than half a millisecond on a 1 GB Ethernet switch.

1. Introduction

Traditional developments in Grid technologies have concentrated on providing batch access to distributed computational and storage resources. The requirements to access, control, and acquire data of widely networked distributed instruments, trigger the need to include scientific equipment such as sensors and probes in the Grid world. This, in turn, raises the need for supporting real-time and reliable interactive work, thus opening a new frontier of research and development in this field.

The GridCC project [21] launched in September 2004 by the European Union addresses these issues. The goal of GridCC is to exploit Grid opportunities for secure and collaborative work of distributed teams to remotely operate and monitor scientific equipments, and to utilize the Grid's massive memory and computing resources for storing and processing data generated by these kinds of equipments.

For this purpose, we are participating in the effort to develop the novel concept of an Instrument Element (IE) [40] that offers a standard Web service interface to integrate instrumentation within the Grid and methods to aggregate different instruments that belong to different Virtual Organization (VO) for achieving a common goal. It provides a simple abstraction of a generic instrument based on services, like the catalogue provision of the instruments controlled by an IE and their description. The remote operations require reliable fast response time with high throughput for controlling the equipments and for consuming the results.

Clearly, the adoption of Web Service (WS) technology as basic building blocks for the instrumentation part of the IE, and in particular the use of SOAP over HTTP, guarantees the interoperability of the implemented services and enables the leveraging of related infrastructure like service discovery [20], security and encryption, and workflow management [22]. However, the modest performance of a Web service-based communication network, limits its use to those cases where high bandwidth and fast response time are not required. In the case of IE, most of the control operations require response time on the order of a fraction of a second. This is achieved by present Web service platforms [7, 39]. However, this response time is not adequate for intercommunication between instruments and for transferring the large quantity of data generated by the equipments; in these cases the bandwidth requirement can be very high [35, 36].

Grid tools [23, 24] for moving files can be used in situations where a file is produced by the permanent

0-7695-3064-8/07 $25.00 © 2007 IEEE
DOI 10.1109/e-Science.2007.26

cache of the IE. During immediate consumption of data or when inter instrument data is exchanged, a fast end-to-end message and/or streaming based communication channel must be established with the peers requiring the data; in addition, since many peers usually require copies of the same data [17], the use of a SOAP-based protocol is clearly not adequate. Instead, a message based system that guaranties high throughput with low latency, and one-to-many data delivery is more appropriate. Fig. 1 provides a detailed description of the mentioned use cases:

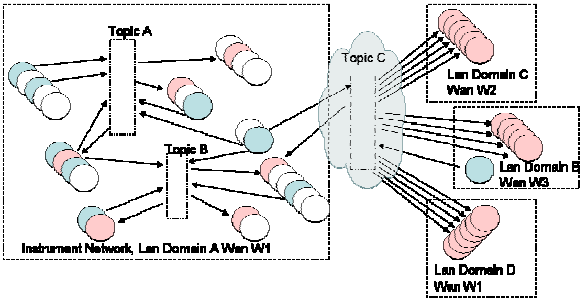

Fig. 1 Typical environment for a grid of instruments

For accomplishing their functionality the instruments need to exchange and filter the generated data using high performance connection network, while at the same time, users around the world want to control the entire system and monitor the on going activity. Performance and scalable architecture are the most important issues in one of our use cases in the field of high energy experiments [35, 36].

To provide a solution for the above requirements and scenario, and for improving performance and usability, we developed RMM-JMS [18], a publish subscribe Java Message Service (JMS) based implementation, on top of our high performance Reliable Multicast Messaging (RMM) layer [19]. This enables the IE to have high-throughput low-latency reliable transport services designed for one-to-many data delivery or many-to-many data exchange in a message-oriented middleware publish/subscribe fashion, which is also JMS compliant. RMM-JMS supports peer-to-peer communication in both brokered and broker-less modes. The broker or bridge is mainly used, whenever we have more than a single multicast domain (e.g., two LANs).

This paper presents and evaluates RMM-JMS and its broker/bridge extension; we compare the performance of our current implementation with the fastest JMS systems in the market [3]. Experimental results show that our system outperforms existing message distribution systems; in particular, a single RMM-JMS node can receive or dispatch more than 900000 messages per second with latency less than half a millisecond while handling data at more than 90MBytes/sec. Under the same conditions top JMS based systems handled less than 5000 messages/sec. (18MBytes/sec).

The rest of this paper is organized as follows: Section 2 presents related work outlining different software architectures that address similar aspects. Section 3 describes RMM-JMS broker/bridge. The experiments and the results obtained for different benchmarks and different systems are presented in Section 4. Finally, in Section 5 we discuss our conclusions and future work.

2. Related Work

Several attempts have been made to integrate heterogeneous high-performance data producers, like instruments, into a complex and distributed framework like the Grid. The majority refer to WS based Service Oriented Architecture (SOA) and to Publish/Subscribe approach. CIMA [1] proposes a common instrument middleware based on Web Services using SOAP over HTTP as a communication layer. JXTA Project [2] attempts to provide a common language (both C++ and java implementation are provided) and platform to be used by all peers. The JXTA environment and language are built around Jxta protocols that are defined via textual representation (i.e., XML) and Jxta pipes. A WS-based standard, WS-Notification [5], describes asynchronous publish/subscribe notification models that can be used for listening to remote service data element updates; WSRF based framework like Apache-WSRF [37] and WSRF.NET [38] use this standard. In RGMA [6], the information resources of a virtual organization (VO) are presented as a single virtual database that contains a set of virtual tables and provides access to this information via a WS interface. The Java Management Extensions (JMX) [8] technology is an open system for management and monitoring; via its instrumentation, agent, and distributed services layers, it can be used for adapting legacy systems, implementing new management tools, and providing monitoring solutions. Jini [9] attempts to provide mechanisms to enable adding, removing, and locating devices and services on the network on top of RMI.

The Java Message Service (JMS) defines a common set of API [33] that allows different peers of a distributed system to communicate in a publish/subscribe message or streaming based way. Several vendors provide JMS implementation in C, C++, C#, Ruby, Perl, Python, and PHP [14, 25, 28] in

order to glue the different parts of a distributed system. Many JMS MessageQueue-based systems, like Naradabrokering [10, 4], JBossMQ [11], JORAM[12], OpenJMS [13], ActiveMQ[14], Arjuna[15], Sun Message Queue [16] FioranoMQ [27] and others [26, 29, 30, 31], have been developed for providing a messaging middleware that allows the interoperation between distributed components of a system. The majority of the implementations utilize a centralized, customizable component that provides the JMS services like publish/subscription and message filtering capabilities. Other implementations like Mantaray [32] utilize a unicast P2P approach. We propose, by implementing RMM-JMS, a P2P multicast approach built on a distributed Reliable Multicast Messaging (RMM) middleware that enables high throughput low latency reliable messaging and streaming distribution.

2.1 Centralized Brokered Message Oriented Middleware

In this type of architecture both the publishers and the subscribers are linked to each other via a centralized component – a broker – that handles topics, message filtering and persistence.

Several implementations provide the possibility to clusterize the broker in order to increase the performance of the entire system. Fig. 2 is an example of such architecture.

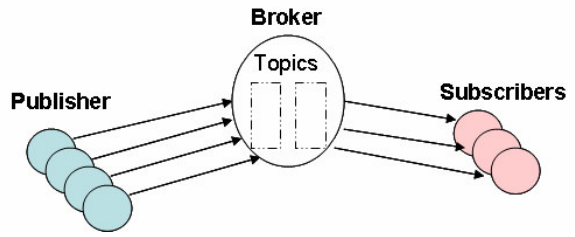

Fig. 2 Centralized (Brokered) Publish Subscribe Architecture

The main advantage of this approach is the possibility to reduce the complexity of the publisher and the subscriber by moving some capabilities to the broker (external entity).

However, this approach introduces a single point of failure that is partially solved with a cluster of brokers. Another drawback is that the peers need to know the broker location; usually this information is provided by an LDAP system or coded into the software. The first solution introduces complexity into the system while the second either reduces the final code portability or introduces additional configuration information that increases the maintenance costs.

2.2 Decentralized Broker-less Message Oriented Middleware

In this approach there is no broker and all the system peers collaborate in order to substitute the functionalities of the broker. This is a typical behaviour of P2P systems, in which actors discover and exchange information with each other (See example of such a system in Fig. 3).

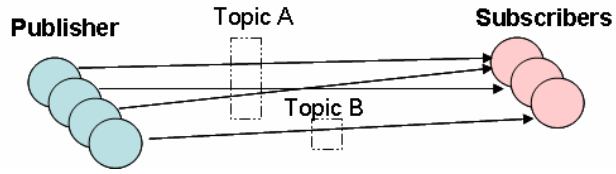

Fig. 3 Broker-less publish subscribe architecture

The main disadvantage of the broker-less approach is that the software that runs in both publishers and subscribers machines is more complex and peers need to "learn" the topology of the network if not initially configured.

Mantaray [32] proposes a solution in which peers discover information related to existing peers subscription and the different available topics via a multicast protocol, while the message delivery communication is done via unicast connections. In JXTA [2] peers find each other and the information related to the existing topics via a pure unicast communication and specific discovery protocols.

Our middleware solution based on RMM-JMS, supports static topology configuration and discovery that is done using unicast, whereas topic filtering, topic mapping and message delivery is done with reliable multicast protocol whenever the underline network supports multicast.

In all multicasts based solutions, communication between peers that live in different NAT domains and behind Firewall requires some gateway bridge that act as a relay.

3. RMM-JMS Broker / HTTP Bridge

RMM-JMS broker/bridge is built on top of RMM protocol that allows hosts to reliably exchange data messages over the standard IP multicast network (in addition to the TCP unicast). RMM exploits the IP multicast infrastructure to ensure scalable resource conservation and timely information distribution with reliability and traffic control added on top of the standard multicast networking. Its services are built as additional network layers on top of UDP/IP and TCP/IP using a NAK based protocol that employs a

fast message to packet mapping. It supports peer-to-peer communication in both brokered and broker-less modes and is singled out by its high performance capabilities.

In a large system, like a Grid, which is composed of a few LAN networks combined with gateways, direct multicast is not supported. In RMM-JMS pub/subs messaging system we implemented a broker/bridge for transferring messages between the gateways and for multicasting when possible.

The messaging broker/bridge, which receives all the publications and subscriptions in the LAN, sends the messages to the appropriate consumers in either multicast or unicast depending on the network configuration. The broker may receive messages from a producer in either unicast or multicast delivery mode.

An important usage of the broker is LAN-WAN-LAN bridging. In such configuration two separate LANs, inside each of which IP multicast is available, are connected via a WAN, where no IP multicast is available. A broker-pair bridge, or broker/bridge for short, is responsible for communicating a multicast (or unicast) message sent in one LAN to a customer in the other LAN. The broker to broker connection is restricted to simple TCP tunnelling but subscriber in each LAN can receive a multicast message sent from the producer (in the local LAN) or a multicast message from the broker (in the remote LAN).

The message producer uses multicast or unicast for publishing. In the unicast mode, the JMS topics are implemented over RMM queues, all with remote end-point at the preconfigured bridge; the bridge's IP address and listening port should be given. No configuration has to be done for multicast publishing and the consumer uses the topic name to figure out which multicast group it has to join.

The broker initially listens for both producers and subscribers. If it gets a publication message for which no subscription has been made, it just drops it. If there are subscribers to this message, it is queued and eventually sent to each of the subscribers. No multicast group is joined until subscriptions are done. Once a topic is subscribed to, the broker joins the group on which this topic may be multicast. When a message with such a topic arrives, it is handled as before. Fig. 4 presents our broker/bridge in a LAN-WAN-LAN setup.

The broker/bridge configuration supports multicast publication in each LAN, where some of the subscribers listen to multicast in another LAN. In such a configuration each bridge forwards to its peer the list of topics for which it has client subscriptions. Thus, only the appropriate topics information will flow between the bridges. In this manner we limit network traffic to the minimum in the slowest link. Moreover we make certain that the bridge does not have to process messages belonging to topics which no client is interested, in a sense, each bridge views its peer as a standard unicast subscriber.

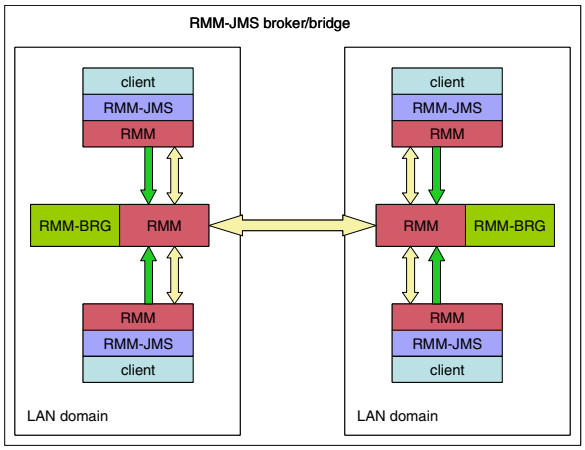

Fig. 4 RMM-JMS broker/bridge in a LAN-WAN-LAN setup

The broker listens on its default unicast address. If so configured, it also joins a range or multicast groups. It creates an RMM set of packet streams for each of the unicast and multicast reception. An important stream is the unicast only broker subscription queue. A typical message on this queue is sent to the JMS client upon creating a JMS topic subscriber. The client should than open an RMM receive queue to receive feedback messages. The two queues are opened on the same connection, which is kept as long as the client's JMS connection is not closed.

When a connection is first created, it is created in a stopped state. This means that publications on this topic will not be sent to the subscriber. To enable message flow, as JMS dictates, the client should send another message on the broker subscription queue, with the directive “start”, and no other data. The flow will be suspended upon sending of a “stop”; a “close” (or reset on the connection) will cancel all subscription associated with the connection. The client can also send “unsubscribe” on a specific topic name.

To support subscriptions, the broker holds a list of all the subscriptions with the queue associated with each one of them. The broker gets all the publications in its stream set receivers, and dispatches the messages to all those subscribers with started connections.

To support the LAN-WAN-LAN bridge, we configure each broker with the address of the peer broker. When the broker starts it behaves as a bridge. Finally, we enable the multicast receivers and transmitters of the brokers, and add the rule that messages received via multicast are sent only using

unicast (no need to repeat messages which are multicast anyway).

4. Experimental Results

In this section we present experimental results of our message distribution system compared with the fastest implementations existing in the market. We performed two different sets of tests; the first set measured the number of requests handled per second in different configurations of different systems, whereas the second set measured the latency introduced by our system.

4.1 The Systems Being Compared

For evaluating our RMM-JMS bridge pub/sub middleware we compared it with the best available systems with similar functionality. The JXTA one-to-one messaging system, that has been exploited in [2] and WS-Notification-based systems, like RGMA, CIMA, and other SOAP based systems built on top of WS technology, were evaluated in [7, 39] and the results can be considered as upper bounds on their performance. The authors of [3, 34] compared different pub/sub message-based systems, with similar functionality to RMM-JMS, and found that the centralized Sun Message Queue [16], and the P2P based Mantaray [32] have the best performance; thus, we compared these two systems with our RMM-JMS.

4.2 Test-bed Hardware and Software

The hardware and software environment of our experiments comprises 32 Dual Xeon 2.40GHz, 1.5GB RAM machines running the CERN Scientific Linux 3.0.4 Operating System, with Kernel 2.4.21-27.0.2.EL.cernsmp and Java 1.4.2_08-b03, linked to each other by a 1 GB Ethernet switch. In this environment we set up a variable number of peers, written in Java, that communicate with each other using the JMS-RMM library, SunMessageQueue3.6 and Mantaray.

4.3 Testing Massage Rate

In this set of tests we compare the effective number of messages that can be injected into each of the systems. We first measure the cost of N to 1 communication (N varies from 1 to 30), where N publishers communicate with one subscriber (Fig. 5a). Next, we evaluate the opposite scenario where one publisher publishes the same message to N (N varies from 1 to 30) clients (Fig. 5b).

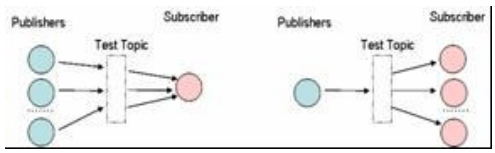

Fig. 5 (5a), (5b): Messages Rate Tests Scenarios

The subscriber in scenario 5a and the publisher in scenario 5b were each running in a dedicated machine; the other subscribers and publishers were uniformly distributed among 30 different machines. Finally, the broker of SunMQ3.6 was installed on an additional dedicated machine. The tests have been repeated varying the payload size of the exchanged messages.

No messages were lost during the tests and the collected statistics on both the publishers' and subscriber's sides showed the same results (per run). Figures 6a, 6b, and 6c present the experimental results, on the subscriber's side, for test configuration 5a with messages of size 100, 1000 and 10000 bytes, respectively. From the figures, we can see that the number of messages handled by the subscriber depends on the messages size and it is independent (with high significance) of the number of publishers. Finally, for the RMM-JMS implementation the total throughput is 91 MBytes/sec when the system exchange messages of 1000 Bytes and it is 75 MBytes/sec in the case of messages of 10000 Bytes. The bundling of messages in RMM and the lack of a broker allow for a better system performance since messages did not have to be routed to an intermediate machine for reaching the subscriber.

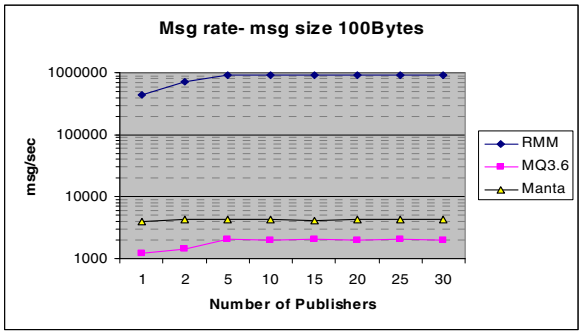

Fig. 6a Message rate for varying number of publishers. Msg size 100 Bytes

The experimental results of test configuration 5b are presented in Figures 7a, 7b, and 7c. Once again, no messages were lost during the tests and the statistics that were computed on both the publisher's and subscribers' sides, showed the same results (per run). The presented measures have been taken from the publisher. As we can see in the plot of RMM the rate is practically constant; this can be explained by the parallelism of the multicast. In contrast, in a standard

P2P and broker-less P2P implementations the rate dropped exponentially with the number of subscribers (we use logarithmic scale for the rate). We also note that a broker-less implementation allow a better system performance because messages are not routed to an intermediate machine in order to reach the subscribers.

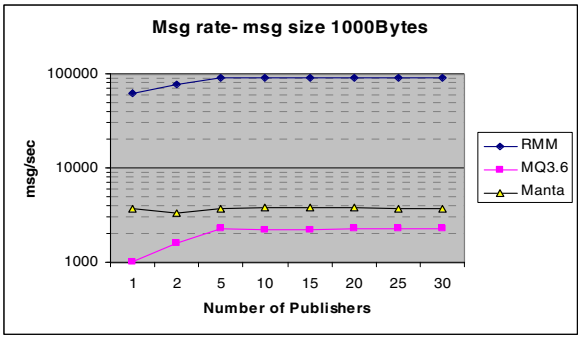

Fig. 6b Message rate for varying number of publishers. Msg size 1000 Bytes

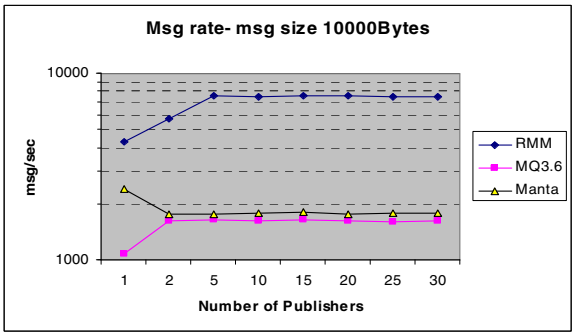

Fig. 6c Message rate for varying number of publishers. Msg size 10000 Bytes

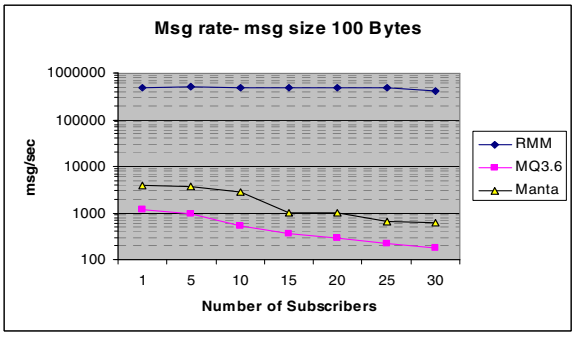

Fig. 7a Message rate for varying number of subscribers. Msg size 100 Byte

We conclude that the multicast system, that is the key feature of RMM, can be used for one-to-many data delivery reaching a transfer rate of 69-80 MBytes/sec per subscriber when the hardware can support it. The maximum message exchange rate is on the order of 550000 messages/second. This number is remarkably high compared with results that were achieved by existing systems [2, 3, 4].

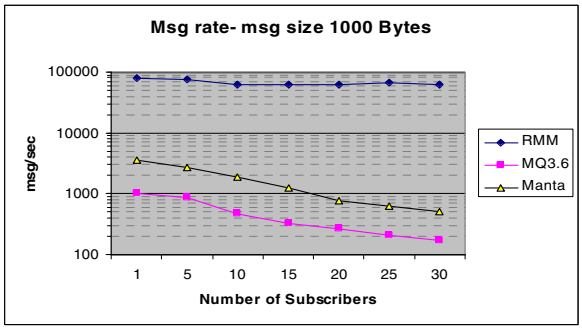

Fig. 7b Message rate for varying number of subscribers. Msg size 1000 Bytes

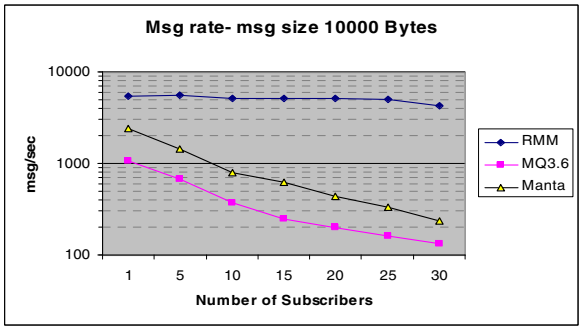

Fig. 7c Message rate for varying number of subscribers. Msg size 10000 Bytes

4.4 Testing Round Trip Time (RTT)

The second set of tests measures the round trip time for a message. In two different machines a publisher sends a message to a given topic and a subscriber was instructed to receive and send back the same message to a different topic.

The publisher was listening for incoming messages. We computed the average round trip time over 1000 samples. The chart in Fig. 8 presents the results for messages of varying size. The results are compared with the time needed for a simple ping between the two machines. As we can see, for messages shorter than 10000 Bytes the experienced RTT is similar for RMM-JMS and SunMQ3.6, while for bigger messages the latency in SunMQ3.6 grows. This behaviour is explained by the additional delay due to brokered communication in SunMQ3.6. The Mantaray implementation introduces 30 ms of minimum delay for aggregating messages and saving time buffering information. Finally, note that the ping measure provides an asymptotic lower bound for such systems. Software overhead in both the sender and receiver, sides can explain the differences compared to RMM-

JMS, this overhead increases consistently with the size of the messages.

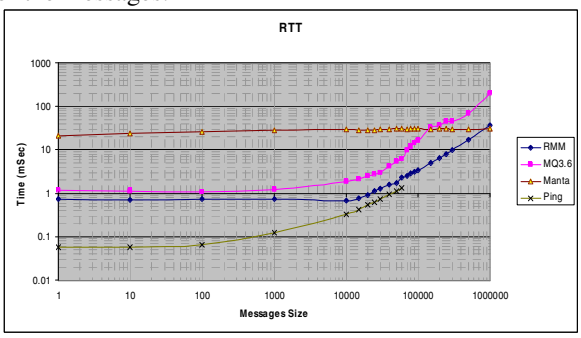

Fig. 8 Round trip time tests, experimental result

4.5 Experiments with a Gateway (Bridge)

In the following set of tests we measured the effective number of messages that can be injected into different systems in two scenarios: the first one (RMM1B) is composed of one publisher that sends messages to a Bridge that dispatches the messages to all the connected subscribers; in the second one (RMM2B) there is one publisher that sends messages to a Bridge (B1) that forwards all its traffic to a second Bridge (B2) whose task is to dispatch the messages to all the connected subscribers. The two scenarios are depicted in Fig. 9.

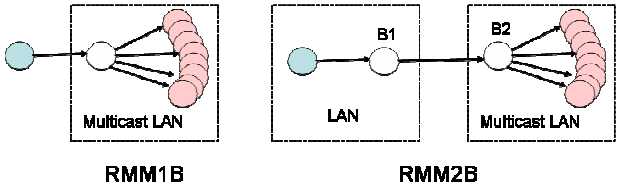

Fig. 9 The two brokered scenarios

The publisher and the subscribers were each running in a dedicated machine. The bridges were installed on two additional dedicated machines. The tests have been repeated varying the payload size of the exchanged messages as in section C.

Figures 10a, 10b, and 10c show in linear scale the message rate on the subscribers' side for both RMM 1B and RMM 2B scenarios. In these figures the values of the message rate are compared with those obtained in the previous test scenario (Section C), i.e. RMM-JMS, Mantaray and Sun Message Queue 3.6.

It is worth to notice that the message rate of both RMM 1B and RMM 2B remain above the rate measured for Mantaray and Sun MQ; this is explained by the small overhead that is introduced to RMM by the broker that act mainly as a relay.

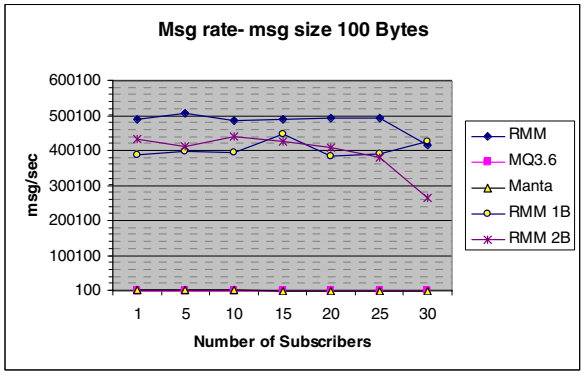

Fig. 10a Message rate for varying number of subscribers. Msg size 100 Bytes

Finally, the rate measured using one Bridge (RMM 1B) is higher than the rate measured using two Bridges (RMM 2B) since the later one has an additional unicast segment.

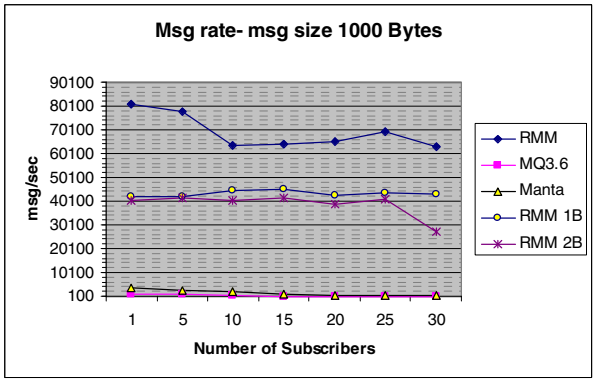

Fig. 10b Message rate for varying number of subscribers. Msg size 1000 Bytes

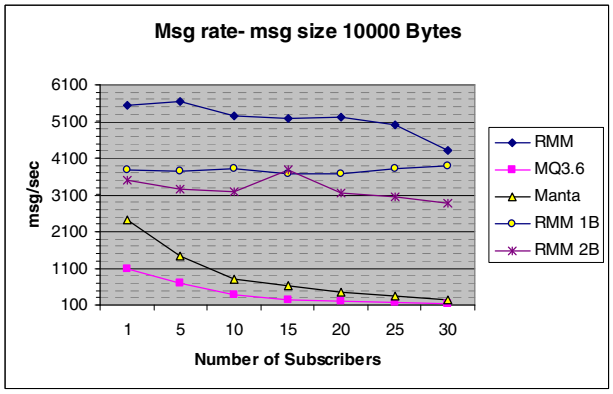

Fig. 10c Message rate for varying number of subscribers. Msg size 10000 Bytes

5. Conclusions

From the results of our experiments, we conclude that in an environment that must support the publication of many messages to many subscribers and

whose size is typical for many instruments, it is better to use a broker-less multicast approach whenever the underlying system supports it.

10. References

[1] D.F. McMullen, T. Devadithya, K. Chiu, "Integrating Instruments and Sensors into the Grid with CIMA Web Services." Proceedings of the Third APAC Conference on Advanced Computing, Grid Applications and e-Research (APAC05). September 25-30, 2005. Gold Coast, Australia.

[2] Daniel Brookshier, Darren Govoni, Navaneeth Krishnan, and Juan Carlos Soto, published JXTA: Java P2P Programming, Sams Publishing March 2002.

[3] Crimson consulting group, High-Performance JMS Messaging A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ: http://www.sun.com/software/products/message_queue/wp_JMSperformance.pdf

[4] Geoffrey Fox and Shrideep Pallickara. "JMS Compliance in the Narada Event Brokering System." Proceedings of the International Conference on Internet Computing (IC-02). June 2002. pp 391-402.

[5] WS Notification specification: also available at http://www-128.ibm.com/developerworks/webservices/library/specification/ws-notification/

[6] R. Byro, B Coghlan, A Cooke, R Cordenonsi, L Cornwall, M Craig, A Djaoui, S Fisher, A Gray, S Hicks, S Kenny, J Leake, O Lyttleton, J Magowan, R Middleton, W Nutt, D O'Callaghan, N Podhorszki, P Taylor, J Walk, A Wilson R-GMA: Production Services for Information and Monitoring in the Grid, AHM2004, Nottingham, UK

[7] A. Slominski, M. Govindaraju, M. R. Head, K. Chiu, M. J. Lewis, R. van Engelen, P. Liu, N Abu-Ghazaleh. A Benchmark Suite for SOAP-based Communication in Grid Web Services. In Proceedings of SC05 (Supercomputing): International Conference for High Performance Computing, Networking, and Storage, Seattle WA, November, 2005.

[8] Vivek Chopra, Amit Bakore, Ben Galbraith, Sing Li, Chanoch Wiggers, Professional Apache Tomcat 5 Wrox May 2004

[9] Jini web Site: http://www.jini.org

[10] The NaradaBrokering Project web Site: http://www.naradabrokering.org/

[11] JBossMQ web Site: http://www.jboss.org/wiki/Wiki.jsp?page=JBossMQ

[12] Joram Project: http://joram.objectweb.org/

[13] OpenJMS Project: http://openjms.sourceforge.net/

[14] ActiveMQ Project: http://www.activemq.org/

[15] Arjuna Project: http://www.arjuna.com/

[16] Sun Message Queue Project: http://www.sun.com/software/products/message_queue/index.xml

[17] Francesco Lelli, Gaetano Maron, Salvatore Orlando and Shlomit Pinter, Bringing instruments into a Grid: an Empiric Approach, WSEAS Transactions on Computers, Vol. 6, No 1, (January 2007), pp. 153-159.

[18] IBM Haifa Research Lab, Reliable Multicast Messaging (RMM) Web site: http://www.haifa.il.ibm.com/projects/software/rmsdk/contact.html

[19] Yoav Tock, Nir Naaman, Avi Harpaz, and Gidon Gershinsky, Hierarchical Clustering of Message Flows in a Multicast Data Dissemination System, Proceedings of the International conference on Parallel and Distributed Computing and Systems (PDCS) November 2005, Phoenix, USA

[20] UDDI Specification, Version 2.0, 3.0, http://www.uddi.org/specification.html. 2005.

[21] GridCC Home Page: www.gridcc.org

[22] Business Process Execution Language for Web Services version also available at:http://www.128.ibm.com/developerworks/library/specification/ws-bpel/

[23] SRM: Storage Management Working Group http://sdm.lbl.gov/srm-wg/

[24] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, GridFTP: Protocol Extensions to FTP for the Grid, update January 2002. Also available at: http://www-fp.globus.org/datagrid/gridftp.html

[25] IBM MQ Series: http://en.wikipedia.org/wiki/MQSeries

[26] elemenope project: http://www.elemenope.org/

[27] Fiorano MQ Project: http://www.fiorano.com/

[28] WebSphere MQ Everyplace project

[29] jtom project: http://www.jtom.de/

[30] Mom4j Project: http://mom4j.sourceforge.net/

[31] SwiftMQ project: http://www.swiftmq.com/

[32] Mantaray Project: http://www.mantamq.org

[33] JMS standard API: http://java.sun.com/products/jms/

[34] Krisoft Solution: JMS Performance Comparison (white paper) also available at http://hosteddocs.ittoolbox.com/krissoft102904.pdf

[35] AGATA Advanced Gamma-Tracking Array design specification: also available at: http://agata.pd.infn.it/Agata-proposal.pdf

[36] Sergio Cittolin, Wesley Smith, Joao Varella, Attila Racz, Michel Della Negra, Alain Herve, CMS TDR 6.2, *The TriDAS Data Acquisition project and High-Level Trigger*, CERN/LHCC, December 2002..

[37] Apache WSRF Project: http://ws.apache.org/wsrf/

[38] WSRF.NET Project: http://www.cs.virginia.edu/~gsw2c/wsrf.net.html.

[39] F. Lelli, G. Maron, and S. Orlando, Improving the Performance of XML Based Technologies by Caching and Reusing Information, Proceeding of the International Conference on Web Services (ICWS) Chicago, September 2006.

[40] E. Frizziero, M. Gulmini, F. Lelli, G. Maron, A. Oh, S. Orlando, A. Petrucci, S. Squizzato and S. Traldi. Instrument Element: A New Grid Component that Enables the Control of Remote Instrumentation. International conference on Cluster Computing and Grid (CCGrid), Singapore May 2006.