Performance evaluation of a fast transport for Data Acquisition

E. Frizziero[2], M. Gulmini[2], Z. Har'El[1], F. Lelli[2], G. Maron[2], P. Molini[2], A. Petrucci[2], S. Pinter[1], S. Squizzato[2], S. Traldi[2]

1 IBM Haifa, Israel, 2 INFN, Laboratori Nazionali di Legnaro, Italy

INTRODUCTION

High-Energy Physics Experiments require more and more fast and reliable transport technologies for Data Acquisition. To this end, possible answers arrive from the industry world, in fact a large amount of technologies based on high-level languages have been developed. We have investigated the Java Message Service (JMS) [1] approach, and in particular a specific implementation of such interface, called Reliable Multicast Messaging (RMM) [2], has been taken in consideration.

TECHNOLOGY DESCRIPTION

JMS is a messaging standard API that allows application components based on Java to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous. There are more than 40 JMS implementations in the market, and many studies that compared these systems have found that the centralized Sun Message Queue [3], and Mantaray [4] - a fully distributed system - have the best performance.

RMM is an implementation of high-throughput low-latency transport services designed for one-to-many data delivery or many-to-many data exchange in a publish/subscribe fashion message-oriented middleware. RMM-JMS provides a JMS implementation on top of RMM that supports both publish/subscribe messaging services and point-to-point services. RMM-JMS is also one of our outcome in the European Funded GRIDCC project.

TESTBED SETUP AND RESULTS

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, Sun Message Queue 3.6 and Mantaray. Our tests have been performed in two different environments: pure peer-to-peer and brokered. Next sections describe these scenarios in details.

Pure peer-to-peer environment

The goal of this set of tests is to see the effective number of messages that can be injected into the system. We first measure the cost of N to 1 communication (N varies from 1 to 30), where N publishers communicate with one subscriber. Next, we evaluate the opposite scenario where one publisher publishes the same message to N (from 1 to 30) clients. Figure 1 describes the test configurations.

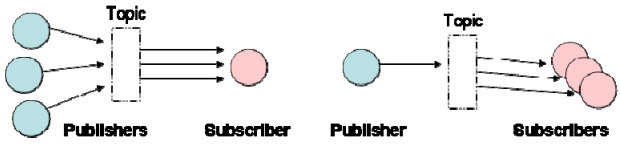

FIG. 1: *(a), (b): Pure peer-to-peer scenarios.*

The subscriber in scenario 1(a) and the publisher in scenario 1(b), were each running in a dedicated machine; the other subscribers and publishers were uniformly distributed among 30 different machines. 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. The payload values were 100 bytes, 1000 bytes and 10000 bytes.

Figure 2 presents the experimental results, on the subscriber's side, for test configuration 1(a) with messages of size 1000 bytes. From the results we can say that the number of messages handled by the subscriber depends on the messages size and from Figure 2 we can see that it is independent (with high significance) from the number of publishers. For the RMM-JMS implementation the maximum 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.

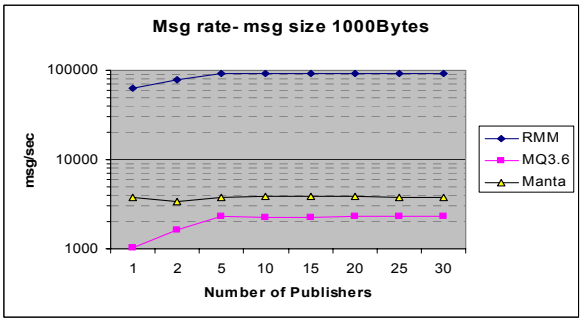

FIG. 2*: message rate with 1000 bytes payload.*

Figure 3 depicts the experimental results of test configuration 1(b) with messages of size 1000 bytes. As we can seen in the chart, the rate of RMM is pretty much 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 also note that a broker-less implementation allow a better system performance because messages do not need to be routed to an intermediate machine in order to reach the subscribers.

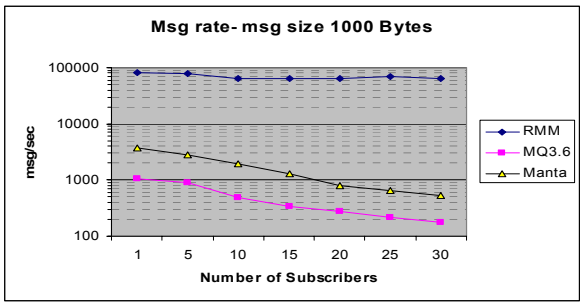

FIG. 3*: message rate with 1000 bytes payload.*

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 [5].

Brokered Environment

In the following set of tests we measured the effective number of messages that can be injected into the different systems in two scenarios (see Figure 4): the first one (RMM1B) is composed by one publisher that sends messages to a broker (or 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.

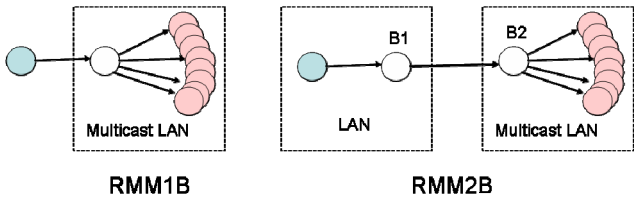

FIG. 4*: The two brokered scenarios.*

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

Figure 5 shows the message rate on subscriber's side for both RMM1B and RMM2B scenarios. Here the maximum throughput is 45 MBytes/sec when the system exchange messages of 1000 Bytes.

There are some important considerations that can be taken observing the chart: firstly the measured message rate is just lower than the rate obtained by a pure peer-to-peer communication; this is due to the fact that we have led into the system a unicast connection to the Bridge, whose activity introduces a delay that causes the rate reduction. In fact the Bridge spends time in receiving the message from an unicast socket and in delivering it using the multicast protocol.

Secondly it is worth to notice that the message rate of both RMM1B and RMM2B remains over and above the rate measured for Mantaray and Sun MQ; the explanation has to be found in the use of RMM that, even with a Bridge, keeps the rate to a considerable high level.

For the same reason, comparing Bridged RMM with Mantaray and Sun MQ implementation, the overall performance does not decrease significantly varying the message size. As described in previous tests, this behavior is mostly due to the use of multicast protocol.

Finally, we can observe that the rate measured using one Bridge (RMM1B) is higher than the rate measured using two Bridges (RMM2B). In this case the first Bridge receives from an unicast channel and delivers on another unicast socket, while the second Bridge act as described before.

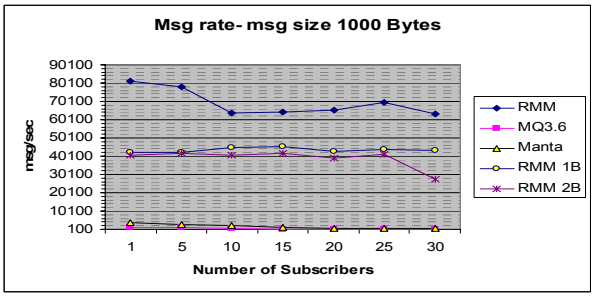

FIG. 5*: Message rate with 1000 bytes payload.*

We conclude that RMM implementation of JMS maintains a very high throughput even if it is used in a brokered scenario; we think that this result could be suitable for a Data Acquisition system that runs on different multicast networks, and that requires an elevate throughput (45 MB/s up to 91 MB/s) to a single machine.

---

[1] JMS standard API: http://java.sun.com/products/jms/
[2] IBM Haifa Research Lab, Reliable Multicast Messaging (RMM) Web site: http://www.haifa.il.ibm.com/projects/software/rmsdk/contact.htm
[3] Sun Message Queue Project: http://www.sun.com/software/products/message_queue
[4] Mantaray Project: http://www.mantamq.org
[5] Crimson consulting group, High-Performance JMS Messaging A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ