@group: Grid Service Monitoring Working Group
@date: 18 Oct 2006
@author: Daniel Rodrigues

@issue: 1st Month Reporting on work
@title: Evaluation of ActiveMQ Broker for the Grid Messaging System.

@acronyms: GSMWG - Grid Service Monitoring Working Group

Introduction

The Grid Service Monitoring Working Group is aiming to create a prototype Messaging System which may increase interoperability, portability, scalability and reliability of the monitoring and measurement of services across the Grid infrastructure. [1]

The present report refers to the first month developments regarding testing of a solution for the messaging infrastructure, based on previous work by Summer Student Wojciech Czech. Work focused on the verification of previous results, and understanding of all the aspects to be weighted in a future implementation.

Context

The confirmation of previous results required studying the different prototype tests already developed. This was done through simulating a test environment as close as possible to the existing one. Please refer to previous reports for more information [2].

Although trying to stick to the existing environment, there was no previous knowledge of all the software pieces in use. Being unknown, the research led to the opportunity of testing under different circumstances. Thus, it allowed to achieve different results before reproducing the exact environment, and led to discovering effects not discussed previously.

Environment Information

Available for the testing were two machines with root access within lxbatch system (lxb6117/lxb6118), as well as the entire lxplus cern system [3], anda personal laptop: pcitgd24. All network connections are [???10/100Mb/1Gb??]
Most systems used were running either SCL4, although some tests within the . [add Memory+CPU description]

.Detailed Description
lxb6117:
/proc/meminfo : total:2074896kB;
/proc/cpuinfo : 2x Intel(R) Xeon(TM) CPU 2.80GHz cache:2048 KB
uname : -o: GNU/Linux -r: 2.6.9-55.0.9.EL.cernsmp -p i686
Usages: broker ActiveMQ4.1; nc producer;
lxb6118:
/proc/meminfo : total:2074896kB;
/proc/cpuinfo : 2x Intel(R) Xeon(TM) CPU 2.80GHz cache:2048 KB
uname : -o: GNU/Linux -r: 2.6.9-55.0.9.EL.cernsmp -p i686

Usages: broker ActiveMQ5.0; nc producer;
pcitgd24:
/proc/meminfo : total:2066644kB;
/proc/cpuinfo : 2x Intel(R) Core(TM)2 CPU T5600 @ 1.83GHz cache: 2048
uname : -o: GNU/Linux -r: 2.6.9-55.0.6.EL.cernsmp -p i686
Usages: nc producer; nc consumer;
lxplus20*:
/proc/meminfo : total:8150040kB;
/proc/cpuinfo : 4x Intel(R) Xeon(TM) CPU 5150 @ 2.66GHz cache:4096 KB
uname : -o: GNU/Linux -r: 2.6.9-55.0.6.EL.cern1smp -p x86_64
Usages: nc producer; nc consumer;
lxplus00*:
/proc/meminfo : total:2054764kB;
/proc/meminfo : 2x Intel(R) Xeon(TM) CPU 2.80GHz cache:512 KB
uname : -o: GNU/Linux -r: 2.4.21-51.EL.cernsmp -p i686
Usages: nc producer; nc consumer;

2 Gbps between lxb6117 and lxb6118;

Work status

We can divide the work time spent during the past month on 3 main chapters:
1. Evaluation of Woijiech tests and past testing environment;
2. Creation of a shell batch testing framework (gman) and interpretation of its results;
3. Fine evaluation of issues arose on 2, and search for proper results;

Following, a brief detail on 2 and 3:

. GMAN testing
After having a first contact with tests already existing, it was decided to concentrate on using STOMP (Streaming Text Orientated Messaging Protocol) as the message protocol. This was mainly due to the simplicity of commands to issue a message: by running netcat on a client, we are enabled to quickly write shell test scripts. OpenWire has not been considered.

GMAN was a marshall of scripts with the aim of: [Scripts will be made available at the wiki pages]
a) automating the tests of a particular broker configuration;
b) easily change the test parameters;
c) retrieve results in a standardized and descriptitive fashion;
d) allow for a statistical analysis of the results, reducing the influence of noise such as network availability, shared processor time, etc.

The core of GMAN was gmanNcTestRun.sh: it requires 5 arguments (payloadSize, expIteration, resultsDirectory, currentNumberProducers, currentNumberConsumers) and accepts 4 optional: (testResultFileName, broker, brokerPort, runComments). In its execution, it creates the payload, adding the proper STOMP headers, and sends a 2^expIteration

number of messages within the same connection to the broker. By use of the time program, it checks how long it takes to send this amount of messages to the broker. It also outputs the messages to the proper file to an xml format (easily converted to csv by use of xsl transformation). Each batch of message is sent to a topic that will be changed according to numberProducers/Subscribers.

On top of this script, there is gmanBatchRun.sh and gmanWasshRun.sh. This are simple scripts with 'for' loops which will automate the testing of different number of Messages and batchPayloadSizes for the first, and different number of producers/subscribers for the later. Unfortunately, netcat was not operational with wassh, therefore there was the need to manually set the necessary number subscribers.

Test runs done using gman were long (some hours for 'large' message sizes, or a 'large' number of messages), and caused some issues, mainly due to the use of lxplus (such as latencies on starting the scripts, or machine reboot/abrupt halt in the remote machine script). However, we will assume this effects to be negligible, as some filtering was done, and later results had more consistency.

Two different types of test will be considered:
a) One run with no consumers, with messageSize varying between 2 KB and 1028 KB, and message number per connection between 2 and 16384;
b) Several runs with varying number of producers/consumers, with messageSize usually varying between 8 and 256 Kb, and number of messages usually between 256 and 4096;

Results were analyzed by converting to csv and using a spreadsheet for plotting. These results are discussed later on this document.

. Fine Tuning test

After the long runs of GMAN, some issues were raised. One of the strange effects we noticed, was that there was clearly an exceptional performance on the beginning of the runs, that would subsequently stabilize into much lower throughputs. Besides, results were clearly apart from the ones previously obtained.

It was decided to try to have a closer look at these issues, and carefully reproduce the previous existing environment. For this fine tuning testing, messages were sent by running netcat non statistically, i.e., manually issuing netcat commands, and carefully looking at the broker status by using jconsole.
To obtain the results discussed later on this document, a iterative approach was used, by increasing the stress on the broker, as well as changing its configuration. Main parameters that had been overlooked were the use of persistency on the server [still trying to clarify it's exact effect and how it works], and the available memory on a specific topic.

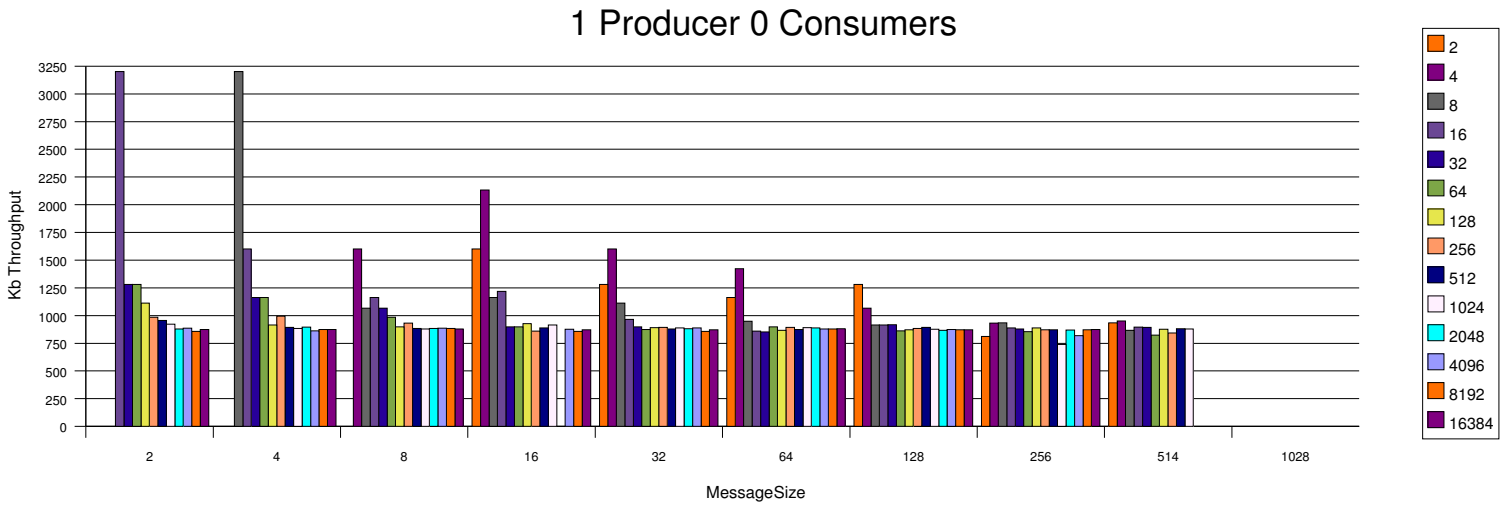
All previous tests had been done with ActiveMQ 4.1.1, the current stable release. Later in the test phase, it was decided to install and have some preliminary results with ActiveMQ5.0, which is the

development version. Results provided a good basis for solving some of the above quoted issues, and discussion on these results also follows.

Test results and discussion

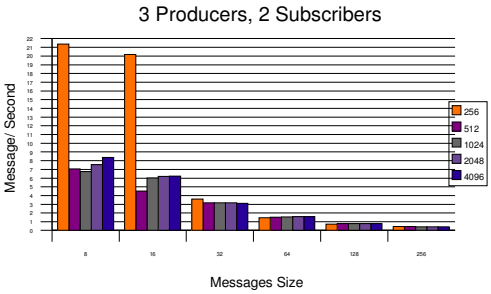
We present two different measurements: Throughput in KB/s and Messages/s where applicable
a) GMAN

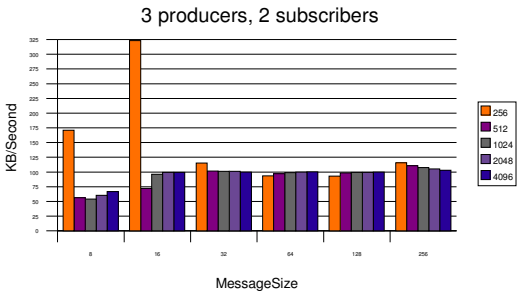
i) Long Run, 1 producer, no consumers.
MessageSize {2,4,8,16,32,64,128,156,514}Kb
NumberMessages {2,4,8,16,32,64,128,256,512,1024,2048,4096,8192,16384}

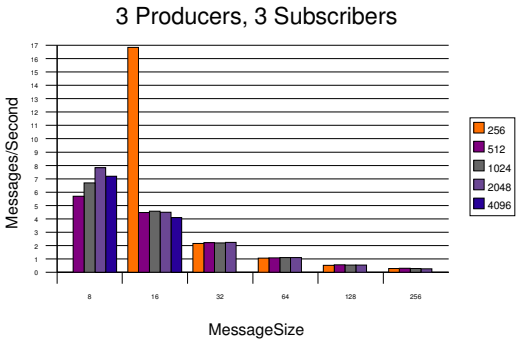


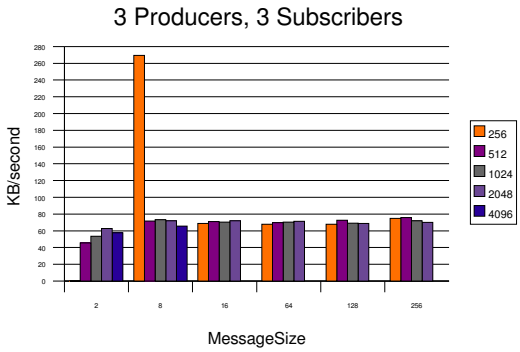
ii) Long Runs, Varying number of Producers and Consumers
MessageSize {8,16,32,64,128,256}Kb
NumberMessages {256,512,1024,2048,4096}

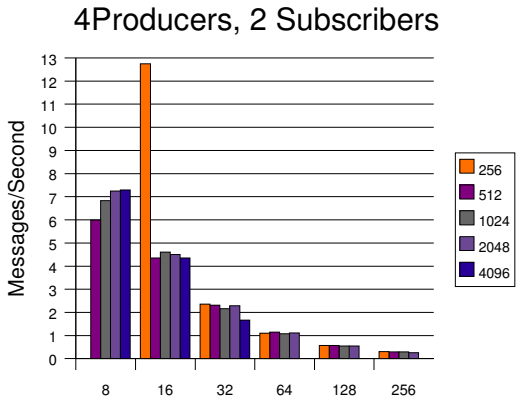
Time is extracted on the average value for the varying number of Producers. Therefore, throughput value shall be calculated by multiplying the given value by the number of Producers.

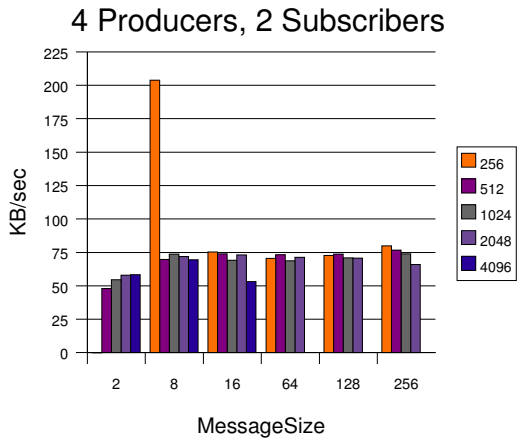

3 Producers, 2 Subscribers
Message/ Second
Messages Size
256
512
1024
2048
4096


3 producers, 2 subscribers
KB/Second
MessageSize
256
512
1024
2048
4096


3 Producers, 3 Subscribers
Messages/Second
MessageSize
256
512
1024
2048
4096

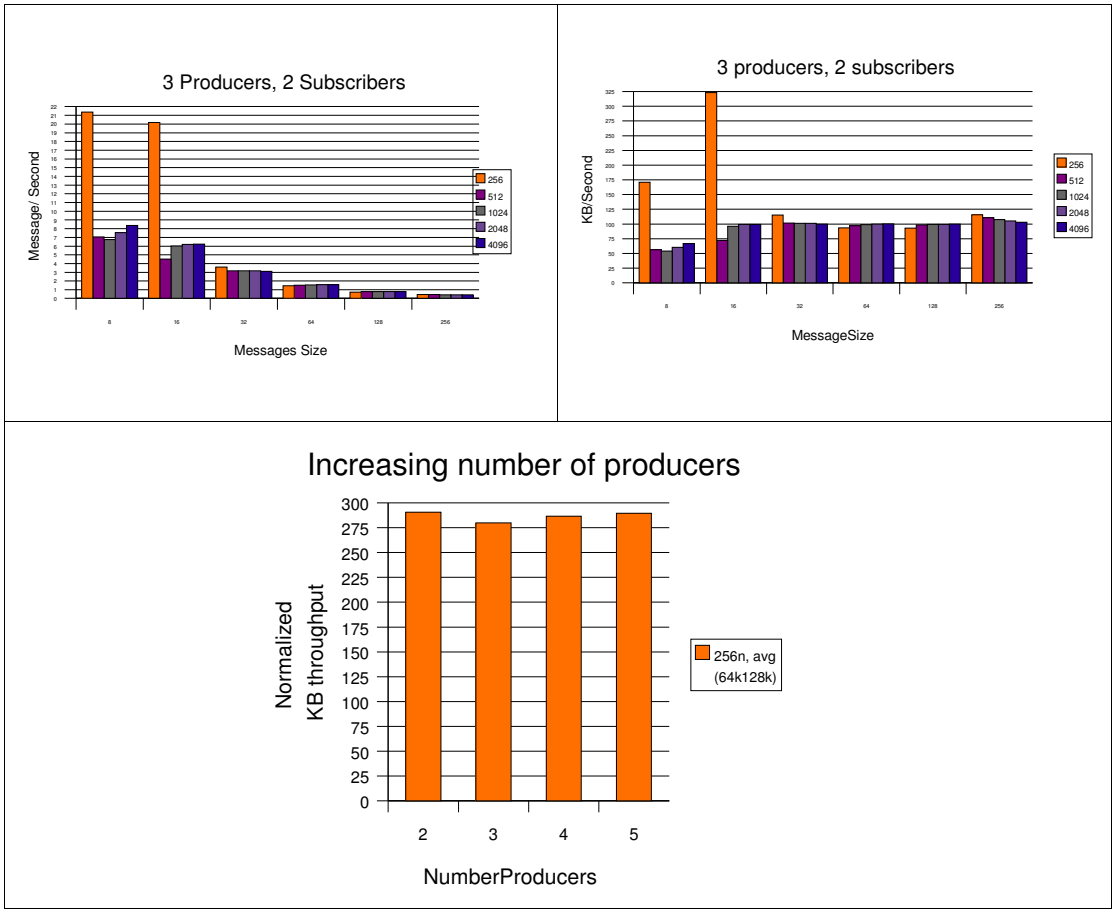

3 Producers, 3 Subscribers
KB/second
MessageSize
256
512
1024
2048
4096


4Producers, 2 Subscribers
Messages/Second
8
16
32
64
128
256
256
512
1024
2048
4096


4 Producers, 2 Subscribers
KB/sec
2
8
16
64
128
256
MessageSize
256
512
1024
2048
4096


5 Producers, 2 Subscribers
Messages/sec
8
16
32
64
128
256
MessageSize
256
512
1024
2048


5 Producers, 2 Subscribers
KB/sec
2
8
16
64
128
256
Message Size
256
512
1024
2048
4096



From this results, we may conclude that
1. There is an effect that should be evaluated when starting the run of the tests;
2. The system scales well for the tested message sizes, Number of messages sent for each connection open, and increasing number of producers, maintaining a constant rate of processing, at a value around 285 KB/sec.

b) fineTuning

Message Size	*Number Messages*	*Speed*	*Saturation*	*Speed after saturation*
1000 B	8192	9102.22 Mes/s [0.9s]	20 cycles	334 Mes/s [~49s]
	16384	8822.15 Mes/s [1.85s]	15 cycles	334 Mes/s [~24.5s]
	32768	8936 Mes/s [3.6s]	6 cycles	334 Mes/s [~98s]
39 B	16384	16384 Mes/s [~1s]	80 cycles	606 Mes/s [~27s]

(in brackets, to complete the command which sends the given number of messages. Saturation occurs after issuing the command the shown number of times.)

According to this results, we were able to see that the peaks seen in the graphs of the first tests were due to a saturation in the configured memory of the broker. Although results are not displayed at the given table, it was concluded that saturation would be more quickly obtained if the available jvm memory for the broker was reduced.

Also interesting enough is that the initial peak approaches the previous results obtained, which leads to the conclusion that tests were done within short runs scope.

Obtained results with ActiveMQ5.0:

Number of Message	*Message Size*	*Speed*
8192	1000 B	819 Mes/s [~10s]
	39 B	2340 Mes/s [~3,5s]

It was observed a constant rate when using ActiveMQ5.0 as the broker. The new queuing paradigm improves the stability of the broker, with a not so bright response for sparse messages, but more reliable and efficient when running it intensively.

Conclusion:

The first tests with ActiveMQ provided a solid start to the problems to be resolved and faced when implementing a future Messaging System for Grid Monitoring.

Previous results were confirmed: ActiveMQ is a reliable and highly scalable messaging provider. Nevertheless, some of the previously considered performance numbers were found to be specific to service start-up conditions.

ActiveMQ5.0 seems to offer a more stable solution, although ActiveMQ could still be considered when looking for a swift response in stochastic and non-intensive use cases.

[1] https://twiki.cern.ch/twiki/bin/view/LCG/GridServiceMonitoringInfo
[2] https://twiki.cern.ch/twiki/bin/view/LCG/GridPublisherDevelopment
[3] http://plus.web.cern.ch/plus/