Towards Benchmarking of AMQP

Stefan Appel
TU Darmstadt, Germany
appel@dvs.tu-darmstadt.de

Kai Sachs
TU Darmstadt, Germany
sachs@dvs.tu-darmstadt.de

Alejandro Buchmann
TU Darmstadt, Germany
buchmann@dvs.tu-darmstadt.de

1. INTRODUCTION

With the increasing importance of event-based systems the performance of underlying event transporting systems, such as message oriented middleware (MOM), becomes business critical. Therefore, we see a strong need for benchmarks for such environments. Several messaging standards and protocols for middleware exist; most popular is the Java Message Service (JMS) which is defining an API rather than a wire protocol. An emerging standard is the new wire level protocol *Advanced Message Queuing Protocol (AMQP)*. It originated in the financial sector and is developed by a consortium of over 20 member organizations.

Within this demo we show a way of evaluating performance of AMQP middleware by using an adapted version of the SPECjms2007 and jms2009-PS benchmarks. It is possible to compare different middleware solutions using JMS as well as AMQP in terms of performance, stability, and scalability.

2. BACKGROUND

2.1 The AMQ Protocol

Advanced Message Queuing Protocol (AMQP) is an increasingly important protocol for MOMs with its origin in financial services industry. The motivation behind AMQP is the need for an open standard which enables complete interoperability between MOM providers [7, 2]. AMQP provides a wire-level protocol specification and not an API as JMS. Due to the popularity of JMS, it was decided to design AMQP to encompass JMS semantics [3] which allows building JMS clients for AMQP products. Therefore, JMS and AMQP complement each other by defining interoperability on the application level (JMS) as well as on the wire level (AMQP). To achieve this interoperability, AMQP specifies an exact semantic of services in its queuing model, which is appropriate for business critical deployments. The specification covers messaging models (P2P, pub/sub, request/response), transaction management, distribution, security, and clustering. In addition AMQP offers several features which are not supported by JMS.

The AMQP specification development process (starting in 2003) is still ongoing and it is expected that AMQP 1.0 will be approved within this year. While the preliminary versions, 0-8 up to 0-10, were build around the concept of Exchanges bound to Queues [3] the terminology changed in the current 1.0 PR2 draft [1]. Messages are exchanged between Nodes via Links whereas Nodes are grouped in Containers interconnected with Sessions. Nodes are for example message producers, consumers or queues. To control message flow and avoid receiving too many messages the receiver has to *issue credit* to a link. By supporting the concepts of durability, (non-)destructive links, and transactional containers, AMQP is capable of providing different Quality of Service (QoS) levels for one-to-one and one-to-many communication.

2.2 AMQP Middleware

Although the AMQP specification is not finalized yet, several products supporting different drafts of AMQP exist today (see Table 1). They are already used in mission critical deployments, e.g., JPMorgan reported an AMQP environment supporting 2,000 users on five continents processing 300 million messages per day [3]. All listed products come with client libraries for different programming language, e.g. Java, C++, Ruby, and Python; [2] demonstrates the installation and use of Qpid along with Python clients. In addition to the listed products JBoss and Apache announced to integrate AMQP support into their popular JMS messaging solutions HornetQ and ActiveMQ; by this a seamless coexistence and integration of JMS and AMQP will become possible.

Vendor	Product	AMQP Ver.
Red Hat	Enterprise MRG	n/a
iMatix Corporation	OpenAMQ	0-9-1
Apache Software Foundation	Qpid	0-10
Rabbit Technologies Ltd.	RabbitMQ	0-9-1
OW2 Consortium	JORAM	0-9

Table 1: AMQP Implementations

2.3 SPECjms2007 and jms2009-PS

SPECjms2007 is a industry standard benchmark developed by the Standard Performance Evaluation Corporation (SPEC) to evaluate JMS middleware using a real-world workload. It is implemented as a Java application framework comprising multiple Java Virtual Machines and threads dis-

DEBS'10, July 12-15, Cambridge, UK.

tributed across a set of client nodes. The benchmark simulates different interactions in a supermarket supply chain with point-to-point and one-to-many communication based upon queues and topics whereas different QoS levels in terms of persistence, durability and transactions are used. A detailed workload description is provided in [5]. Additionally, to evaluate different QoS settings and queue/topic configurations more fine granular, jms2009-PS was developed based upon SPECjms2007 [4].

3. BENCHMARKING AMQP

To benchmark AMQP, we need a workload scenario and a benchmark framework. Since AMQP and JMS are used in similar business cases, the SPECjms2007 standardized workload is used to build our AMQP benchmark on top of it. Using a standard workload allows a comparison of different products but also helps developers to identify weaknesses and to give design guidelines. Further, by having insights into system utilization under certain loads, the scalability and influence of tuning parameters can be evaluated. A benefit of our approach is, that it allows to compare middleware products supporting different standards such as AMQP and JMS directly.

3.1 Implementation

To allow benchmarking of AMQP middleware using the SPECjms2007 workload an interface was added to the benchmark frameworks of SPECjms2007 and jms2009-PS. On the client side SPECjms2007 and jms2009-PS were adapted to use the Qpid Java client as an interface. This is accomplished by providing a JMS Connection Factory accessible via a Java Naming and Directory Interface (JNDI) lookup. The Qpid client translates the JMS requests and uses AMQP for the communication with the server. Further, we had to adjust the benchmark frameworks in a performance neutral way without violating the JMS specification. The result is a framework which can be configured easily to communicate with pure JMS as well as AMQP middleware.

3.2 Demonstration: Benchmarking Qpid

The prototypic test setup for benchmarking AMQP is based upon Qpid and the Qpid Java client which supports JMS. Figure 1 shows the basic components involved: the server is running the Qpid C++ broker with a persistence module loaded (persistence is required by SPECjms2007).

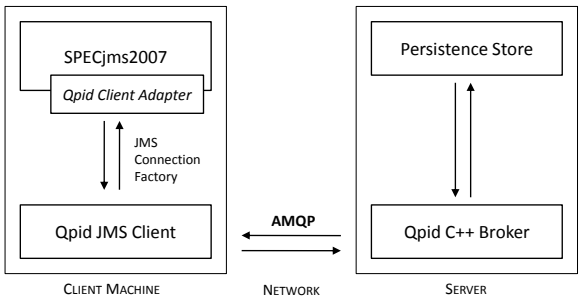

Figure 1: Qpid Test Scenario

3.3 Related Work

Since AMQP is an emerging standard it has not been in focus of research yet. In terms of benchmarking, one work addresses the performance of AMQP using Infiniband [6]. The researchers chose Qpid as AMQP middleware and did not enable persistence; by this network I/O becomes the bottleneck rather than disk I/O. Five simple benchmarks were used to evaluate different performance aspects while SPECjms2007 uses a complex workload originating from a real-world scenario.

4. SUMMARY

AMQP is expected to gain momentum within the next years enabling the seamless integration of different MOM solutions. For this, evaluating and comparing the performance of different MOM is essential in order to reliably transport messages while minimizing the costs.

This demo paper presents a way of evaluating performance of AMQP supporting middleware; in addition a direct comparison with JMS middleware is possible. This is accomplished by adapting SPECjms2007 to utilize the Qpid Java client; the setup was tested successfully with different Qpid versions.

In addition to pure evaluation of performance, e.g. in terms of resource needs per message, the proposed approach is helpful for MOM developers, too. The benchmark utilizes a variety of different communication parameters and allows testing for reliability and robustness of implementations. Finally, scalability can be evaluated as well together with fail over capabilities in complex cluster setups.

For application developers and MOM administrators the use of jms2009-PS allows evaluating different configurations, e.g.: message producers can send messages to dedicated destinations per message type or follow a more generic approach and send all messages to a single destination pursuing the idea of an *enterprise service bus (ESB)*. While the use of multiple destinations inhibits expandability, using just one destination increases filter complexity on consumer side; the respective influence on performance can be evaluated using the presented benchmarking approach.

5. REFERENCES

[1] AMQP Working Group. Advanced message queuing protocol, 2010. `http://www.amqp.org`.

[2] J. Kramer. Advanced Message Queuing Protocol (AMQP). *Linux J.*, 2009(187):3, 2009.

[3] J. O'Hara. Toward a commodity enterprise middleware. *Queue*, 5(4):48–55, 2007.

[4] K. Sachs, S. Appel, S. Kounev, and A. Buchmann. Benchmarking publish/subscribe-based messaging systems. In *Database Systems for Advanced Applications: DASFAA 2010 International Workshops: BenchmarX'10*, LNCS. Springer-Verlag, 2010.

[5] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann. Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. *Performance Evaluation*, 66(8):410–434, Aug. 2009.

[6] H. Subramoni, G. Marsh, S. Narravula, P. Lai, and D. Panda. Design and evaluation of benchmarks for financial applications using advanced message queuing protocol (amqp) over infiniband. *High Performance Computational Finance, 2008. WHPCF 2008. Workshop on*, pages 1–8, nov. 2008.

[7] S. Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6):87–89, 2006.