Benchmarking Publish/Subscribe-based Messaging Systems

Kai Sachs[1], Stefan Appel[1], Samuel Kounev[2], and Alejandro Buchmann[1]

[1] Databases and Distributed System Group, TU Darmstadt, Germany
lastname@dvs.tu-darmstadt.de
[2] Descartes Research Group, Karlsruhe Institute of Technology
skounev@acm.com

Abstract. Publish/subscribe-based messaging systems are used increasingly often as a communication mechanism in data-oriented web applications. Such applications often pose serious performance and scalability challenges. To address these challenges, it is important that systems are tested using benchmarks to evaluate their performance and scalability before they are put into production. In this paper, we present *jms2009-PS*, a new benchmark for publish/subscribe-based messaging systems built on top of the SPECjms2007 standard workload. We introduce the benchmark and discuss its configuration parameters showing how the workload can be customized to evaluate various aspects of publish/subscribe communication. Finally, we present a case study illustrating how the benchmark can be used for performance analysis of messaging servers.

1 Introduction

Publish/subscribe-based messaging systems are used increasingly often as a communication mechanism in data-oriented web applications such as Web 2.0 applications, social networks, online auctions and information dissemination applications to name just a few [1]. Moreover, the publish/subscribe paradigm is part of major technology domains including Enterprise Service Bus, Enterprise Application Integration, Service-Oriented Architecture and Event-Driven Architecture. With the growing adoption of these technologies and applications, the need for benchmarks and performance evaluation tools in the area of publish/subscribe systems increases. While general benchmarks for message-oriented middleware (MOM) exist, no benchmarks specifically targeted at publish/subscribe communication have been proposed. In this paper, we present a new benchmark for publish/subscribe-based messaging systems built on top of the SPECjms2007 standard workload.

SPECjms2007 is the current industry-standard benchmark for MOM servers based on the JMS (Java Message Service) standard interface [2]. It was developed by the Java subcommittee of the Standard Performance Evaluation Corporation (SPEC) with the participation of TU Darmstadt, IBM, Sun, BEA, Sybase, Apache, Oracle and JBoss. One of the major benefits of SPECjms2007 is that, in

addition to providing a standard workload and metrics for MOM performance, the benchmark provides a flexible and robust framework for in-depth performance evaluation of messaging infrastructures. It allows to create custom workload scenarios and interactions to stress selected aspects of the MOM infrastructure. Examples of such user-defined scenarios can be found in [3] and [4]. While SPECjms2007 includes some limited publish/subscribe communication as part of the workload, the focus of the benchmark is on point-to-point (PtP) communication via queues which dominate the overall system workload [5]. Moreover, the workload does not exercise message filtering through JMS selectors which is an important feature of publish/subscribe messaging that typically causes the most performance and scalability issues.

To address the need for a workload focused on publish/subscribe messaging, we developed the new *jms2009-PS* benchmark which uses the SPECjms2007 workload as basis. A preliminary version of the benchmark was demonstrated at the SIGMETRICS/Performance 2009 Demo Competition [6]. In this paper, we introduce the benchmark and discuss its configuration parameters showing how the workload can be customized to evaluate different aspects of publish/subscribe communication. Overall, jms2009-PS provides more than 80 new configuration parameters allowing the user to customize the workload in terms of the number of topics, the number of subscriptions, the number and type of selectors, and the message delivery modes. After discussing the configuration parameters, we present a case study, in which we demonstrate how to use jms2009-PS for evaluating alternative ways of implementing publish/subscribe communication in terms of their overhead, performance and scalability.

The rest of this paper is structured as follows: We start with some background on message-oriented middleware and the SPECjms2007 benchmark in Section 2. Following this, we present the jms2009-PS benchmark in Section 3. We introduce the various configuration parameters and show how the workload can be customized. Finally, in Section 4, we present our case study and wrap up with some concluding remarks in Section 5.

2 Background

2.1 Message-Oriented Middleware

Message-oriented middleware (MOM) is a specific class of middleware that supports loosely coupled communication among distributed software components by means of asynchronous message-passing as opposed to a request/response metaphor. The loose coupling of communicating parties has several important advantages: i) message producers and consumers do not need to know about each other, ii) they do not need to be active at the same time to exchange information, iii) they are not blocked when sending or receiving messages [7].

The Java Message Service (JMS) [2] is a standard Java-based interface for accessing the facilities of enterprise MOM servers. JMS supports two messaging models: *point-to-point (PtP)* and *publish/subscribe (pub/sub)*. With PtP messaging each message is sent to a specific *queue* and is retrieved and processed

by a single consumer whereas with pub/sub messaging each message is sent to a specific *topic* and it may be delivered to multiple consumers interested in the topic. Consumers are required to register by subscribing to the topic before they can start receiving messages. In the pub/sub domain, message producers are referred to as *publishers* and message consumers as *subscribers*. JMS queues and topics are commonly referred to as *destinations*. The two messaging models are depicted in Figures 1 and 2. The JMS specification defines several modes of message delivery with different quality-of-service attributes:

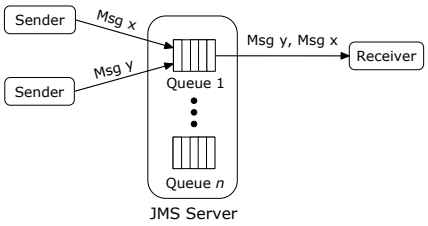


Fig. 1. Point-to-point messaging.

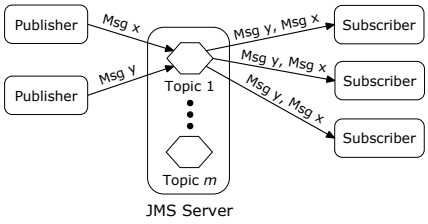


Fig. 2. Pub/sub messaging.

Non-Persistent/Persistent: In non-persistent mode, pending messages are kept in main memory buffers while they are waiting to be delivered and are not logged to stable storage. In persistent mode, the JMS provider takes extra care to ensure that no messages are lost in case of a server crash. This is achieved by logging messages to persistent storage such as a database or a file system.

Non-Durable/Durable: JMS supports two types of subscriptions, durable and non-durable. With non-durable subscriptions a subscriber will only receive messages that are published while he is active. In contrast to this, durable subscriptions ensure that a subscriber does not miss any messages during periods of inactivity.

Non-Transactional/Transactional: A JMS messaging session can be transactional or non-transactional. A transaction is a set of messaging operations that are executed as an atomic unit of work.

In addition to the above described delivery modes, JMS allows the specification of *selectors* to enable message filtering. When publishing messages, producers can specify property-value pairs (e.g., *"color=red"*) which are stored in the message headers. When subscribing, consumers can specify a selector to receive only messages with certain property values (e.g., *"color=blue AND size=42"*). Selectors are specified using a subset of the SQL92 conditional expression syntax. For a more detailed introduction to MOM and JMS the reader is referred to [8, 2].

2.2 SPECjms2007

The SPECjms2007 benchmark models a supermarket supply chain where RFID technology is used to track the flow of goods. The participants involved are the headquarters (HQ) of the supermarket company, its stores (SM), its distribution centers (DC) and its suppliers (SP). SPECjms2007 defines seven interactions between the participants in the scenario:

1. Order/shipment handling between SM and DC
2. Order/shipment handling between DC and SP
3. Price updates sent from HQ to SMs
4. Inventory management inside SMs
5. Sales statistics sent from SMs to HQ
6. New product announcements sent from HQ to SMs
7. Credit card hot lists sent from HQ to SMs

Interactions 1 and 2 represent a chain of messages while the rest of the interactions include a single message exchange [4]. A single parameter called `BASE` determines the rate at which interactions are executed and is used as a scaling factor. The benchmark is implemented as a Java application comprising multiple JVMs and threads distributed across a set of *client nodes*. For every destination (queue or topic), there is a separate Java class called *Event Handler (EH)* that encapsulates the application logic executed to process messages sent to that destination. Event handlers register as listeners for the queue/topic and receive call backs from the messaging infrastructure as new messages arrive. In addition, for every physical location, a set of threads (referred to as *driver threads*) is launched to drive the benchmark interactions that are logically started at that location.

2.3 Related Work

Over the last decade several proprietary and open-source benchmarks for evaluating MOM platforms have been developed and used in the academia and industry including SonicMQ' Test Harness [9], IBM's Performance Harness for Java Message Service [10], Apache's ActiveMQ JMeter Performance Test [11] and JBoss' Messaging Performance Framework [12]. Using these and other similar benchmarks, numerous performance studies have been conducted and published, see for example [13–20]. While the benchmarks we mentioned have been employed extensively for performance testing and system analyses, unfortunately, they use artificial workloads that do not reflect any real-world application scenario. Furthermore, they typically concentrate on stressing individual MOM features in isolation and do not provide a comprehensive and representative workload for evaluating the overall MOM server performance. For a more detailed discussion of related work we refer the interested reader to [4, 21].

Table 1. Configuration parameters supported for each message type.

Intr.	Message	Location	T	P	D	Q	TD	ST	Description
1	order	DC	✓	✓	✓	✓	✓	-	Order sent from SM to DC.
	orderConf	SM	✓	✓	✓	✓	✓	-	Order confirmation sent from DC to SM.
	shipDep	DC	✓	✓	✓	✓	✓	-	Shipment registered by RFID readers upon leaving DC.
	statInfo-OrderDC	HQ	✓	✓	✓	✓	✓	-	Sales statistics sent from DC to HQ.
	shipInfo	SM	✓	✓	✓	✓	✓	-	Shipment from DC registered by RFID readers upon arrival at SM.
	shipConf	DC	✓	✓	✓	✓	✓	-	Shipment confirmation sent from SM to DC.
2	callForOffers	HQ	✓	✓	✓	-	✓	✓	Call for offers sent from DC to SPs (XML).
	offer	DC	✓	✓	✓	✓	✓	-	Offer sent from SP to DC (XML).
	pOrder	SP	✓	✓	✓	✓	✓	-	Order sent from DC to SP (XML).
	pOrderConf	DC	✓	✓	✓	✓	✓	-	Order confirmation sent from SP to DC (XML).
	invoice	HQ	✓	✓	✓	✓	✓	-	Order invoice sent from SP to HQ (XML).
	pShipInfo	DC	✓	✓	✓	✓	✓	-	Shipment from SP registered by RFID readers upon arrival at DC.
	pShipConf	SP	✓	✓	✓	✓	✓	-	Shipment confirmation sent from DC to SP (XML).
	statInfo-ShipDC	HQ	✓	✓	✓	✓	✓	-	Purchase statistics sent from DC to HQ.
3	priceUpdate	HQ	✓	✓	✓	-	✓	-	Price update sent from HQ to SMs.
4	inventoryInfo	SM	✓	✓	✓	✓	✓	-	Item movement registered by RFID readers in the warehouse of SM.
5	statInfoSM	HQ	✓	✓	✓	✓	✓	-	Sales statistics sent from SM to HQ.
6	product-Announcement	HQ	✓	✓	✓	-	✓	-	New product announcements sent from HQ to SMs.
7	creditCardHL	HQ	✓	✓	✓	-	✓	-	Credit card hotlist sent from HQ to SMs.

3 jms2009-PS - A Pub/Sub Benchmark

We now present the new *jms2009-PS* benchmark which is specifically targeted at pub/sub systems. We developed jms2009-PS using the SPECjms2007[4] workload and its scaling strategy as a basis [22]. Overall, we added more than 80 new configuration parameters allowing the user to customize the workload to his needs. All configurations are identical in terms of the number of subscriptions and the message throughput generated for a given scaling factor, however, they differ in six important points:

1. number of topics and queues used
2. number of transactional vs. non-transactional messages
3. number of persistent vs. non-persistent messages
4. total traffic per topic and queue
5. complexity of used selectors (filter statements)
6. number of subscribers per topic

While the benchmark is targeted at pub/sub workloads, it allows to use queue-based PtP messaging in cases where messages are sent to a single con-

sumer. This allows to compare the costs of queue-based vs. topic-based communication for different message delivery modes. In the case of topic-based communication, for each interaction several implementations are supported. In the first implementation, all types of messages are exchanged using one common topic per interaction. Each message consumer (e.g., orders department in DC1) subscribes to this topic using a selector specifying two filters that define the messages he is interested in: message type (e.g., orders) and location ID (e.g., DC 1). The message type and location ID are assigned as properties of each message published as part of the respective interaction. In the second implementation, a separate topic is used for each type of message (e.g., one topic for orders, one for invoices). Consequently, message consumers do not have to specify the message type at subscription time, but only their location ID. It is easy to see that the number of subscribers per topic is lower and the filtering is simpler (only one property to check) in the second implementation compared to the first one. In the first implementation, more traffic is generated per topic, while in the second implementation the traffic per topic is less but the system has to handle more topics in parallel. Therefore, the two implementations stress the system in different ways and allow to evaluate different performance aspects. In addition to these two implementations, the benchmark supports several further implementations which allow to stress additional aspects of topic-based communication. The user can select an implementation by means of the Target Destination (TD) parameter discussed in the next section.

3.1 Configuration Parameters

In this section, we describe in detail the new configuration parameters introduced in jms2009-PS. The parameters can be configured on a per message type basis. Table 1 shows the parameters supported for each message type. In the following, we briefly describe each parameter.

Transactional [*true*|*false*] (T) Specifies whether messages should be sent as part of a transaction.

Persistent [*true*|*false*] (P) Specifies whether messages should be sent in persistent mode.

Durable [*true*|*false*] (D) Specifies whether a durable subscription should be used by message consumers.

Queue [*true*|*false*] (Q) Specifies whether a queue or a topic should be used in cases where there is a single message consumer.

Target Destination (TD) Specifies for each message type the set of topics and respective selectors that should be used to distribute messages to the target consumers. The benchmark supports six different target destination options. Depending on the selected configuration, it automatically takes care of configuring message properties (set by producers) and selectors (set by consumers at subscription time) to guarantee that messages are delivered to the correct consumers. The target destination options supported by jms2009-PS are shown in Table 2. For each option, the set of topics and the required selectors are described.

Table 2. Target destination options.

Setting	Description	Selector
LocationID-MessageType	A separate topic for each combination of location instance and message type is used, e.g., a topic per DC for order messages: `DC1_OrderT` for DC 1, `DC2_OrderT` for DC 2, etc.	– No selectors are needed.
MessageType	A single topic per message type is used, e.g., a topic `DC_OrderT` for order messages of all DCs.	– TargetLocationID= '*locationID*'
Interaction	A single topic per interaction is used, e.g., a topic `Interaction1_T` for all messages involved in Interaction 1.	– TargetLocationID= '*locationID*' – MessageType= '*messageType*'
LocationType	A single topic per location type is used, e.g., a topic `SM_T` for all messages sent to SMs.	– *TargetLocationID*= '*locationID*' – MessageType= '*messageType*'
LocationID	A separate topic for each location instance is used, e.g., a topic `SM1_T` for all messages sent to SM 1.	– MessageType= '*messageType*'
Central	One central topic for all messages is used, e.g., one topic `T` for all messages that are part of the seven interactions.	– LocationType= '*locationType*' – TargetLocationID= '*locationID*' – MessageType= '*messageType*'

Subscription Type [*IN*|*OR*|*SET*] (ST) In Interaction 2, a distribution center (DC) sends a `CallForOffers` to suppliers (SP). Each SP offers a subset of all product families and is only interested in the `CallForOffers` messages targeted at the respective product families. There are multiple ways to implement this communication pattern and jms2009-PS supports the following options:

- **Use a separate topic for each product family**: The SP has to subscribe to all topics corresponding to the product families he is interested in and no selector is needed.
- **Use one topic for all product families**: The SP has to subscribe to this topic using a selector to specify the product families he is interested in. jms2009-PS offers three ways to define the respective subscription:
  - **Using multiple OR operators**: The SP places a single subscription using the following selector: *ProductFamily*="PF1" `OR` *ProductFamily*="PF2" `OR` ... `OR` *ProductFamily*="PFn"
  - **Using a single IN operator**: The SP places a single subscription using the following selector: *ProductFamily* `IN` ("PF1","PF2",...,"PFn")
  - **Using a set of subscriptions**: The SP subscribes for each product family he is interested in separately:
    ProductFamily="PF1" [$\cdots$] *ProductFamily*="PFn"

4 Case Study

4.1 Introduction

We now present a case study illustrating how jms2009-PS can be used for performance analysis of messaging servers. The environment in which we conducted our case study is depicted in Figure 3. ActiveMQ server was used as a JMS server installed on a machine with two quad-core CPUs and 16 GB of main memory. The server was run in a 64-bit JRockit 1.6 JVM with 8 GB of heap space. A RAID 0 disk array comprised of four disk drives was used for maximum performance. ActiveMQ was configured to use a file-based store for persistent messages with a 3.8 GB message buffer. The jms2009-PS drivers were distributed across three machines. To further increase the network capacity, a separate GBit link was installed between the JMS server and the third driver machine. The latter was configured to always use this link when accessing the server. The drivers were distributed across the machines in such a way that the network traffic was load-balanced between the two networks.

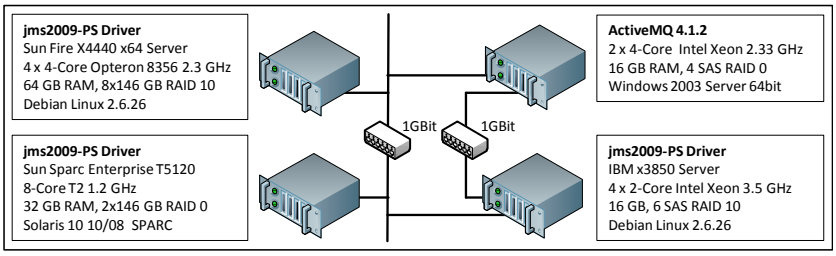


Fig. 3. Experimental environment.

4.2 Test Scenarios

We studied three different scenarios which were identical in terms of the total number of messages sent and received for a given scaling factor (`BASE`). Transactions and persistent message delivery were configured as defined in the SPECjms2007 workload description[4]. The scenarios differ in the number of message destinations and destination types used for communication. Figure 4 illustrates the configurations used in the three scenarios for two of the message types: `order` messages sent from SMs to DCs and `orderConf` messages sent from DCs to SMs (cf. Table 1).

- **Scenario I (SPECjms2007-like Workload):** The workload is configured similar to the SPECjms2007 workload, i.e., it uses mainly queues for communication. Each location instance has its own queue for each message type and therefore there is no need for selectors.

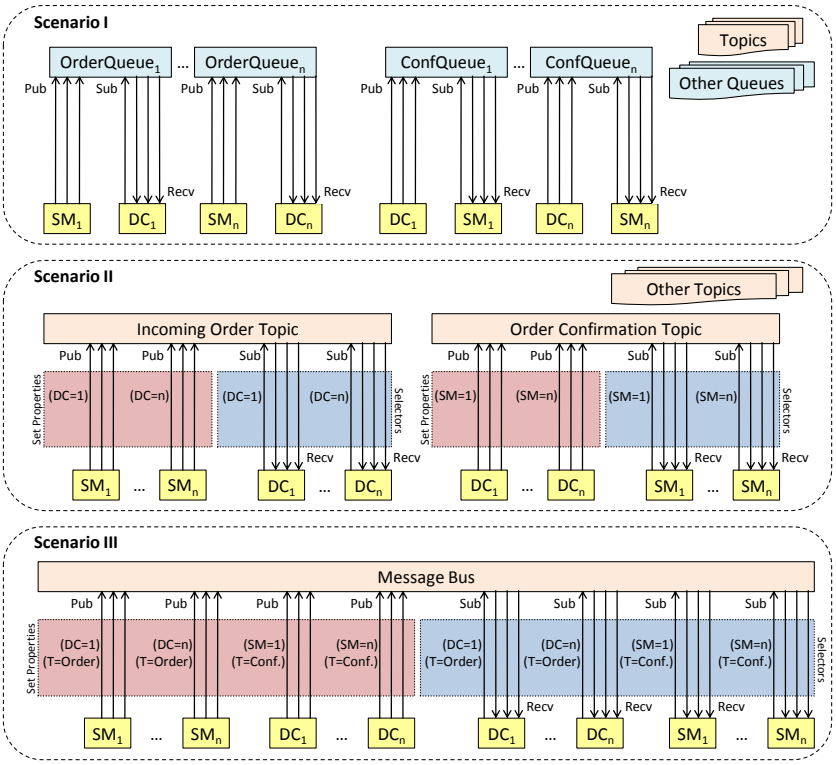
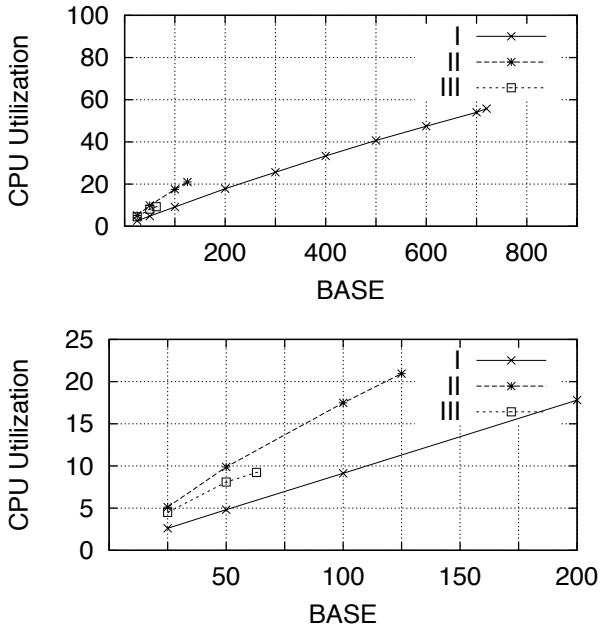


Fig. 4. Considered Scenarios

– **Scenario II (Pub/Sub with Multiple Topics):** For each message type, a separate topic is used, i.e., the TD configuration parameter is set to `MessageType` (cf. Table 2).
– **Scenario III (Pub/Sub with Message Bus):** One topic is used for all messages, i.e., the TD configuration parameter is set to `Central` (cf. Table 2).

The three scenarios differ mainly in terms of the flexibility they provide. While Scenario I is easy to implement given that no properties or selectors are necessary, it requires a reconfiguration of the MOM server for each new location or message type since new queues have to be set up. In contrast, Scenarios II and III, which only use topics, provide more flexibility. In Scenario II, a reconfiguration of the MOM server is necessary only when introducing new message types. Scenario III doesn't require reconfiguration at all since a single topic (message bus) is used for communication. In addition, Scenarios II and III support one-to-many communication while the queue-based interactions in Scenario I are limited to one-to-one communication. One-to-many communication based on pub/sub allows to easily add additional message consumers, e.g., to maintain statistics about orders. On the other hand, the use of a limited number of topics in Scenarios II and III degrades the system scalability. As shown in the next sec-

tion, the jms2009-PS benchmark allows to evaluate the trade-offs that different configurations provide in terms of flexibility, performance and scalability.



Scenario	Max Load	CPU/BASE	Avg. Dlv. Latency (ms)
I	720	0.077	123
II	125	0.168	1587
III	63	0.147	3235

Fig. 5. Experimental Results

4.3 Experimental Results

Figure 5 presents the experimental results for the three scenarios described above. It shows the CPU utilization for increasing workload intensities (`BASE`), the maximum load that can be sustained by each scenario, the CPU time per unit of the BASE parameter and the average message delivery latency. The results show the scalability and performance of the three configurations as well as their costs in terms of CPU consumption. Scenario I scales up to BASE 720 and exhibits the lowest message delivery latency ($123ms$). The flexibility provided by Scenario II and III comes at the cost of much worse scalability and performance. The maximum load that can be sustained in Scenario II and Scenario III is respectively 6 and 12 times lower than that in Scenario I. Similarly, the average message delivery latency is about 13 times higher for Scenario II compared to Scenario I and about 26 times higher for Scenario III. Thus, the flexibility

provided by Scenario II and III comes at a high price. This is due to two reasons: i) the use of selectors leads to roughly two times higher CPU processing time per message as shown in Figure 5, ii) the use of topics for communication leads to synchronization delays. Comparing Scenarios II and III reveals that the selector complexity in this case does not have a significant impact on the CPU processing time per message. What is much more significant is the number of topics used for communication. The single topic in Scenario III clearly leads to a scalability bottleneck and explosion of the message delivery latency. In the third scenario, the throughput was limited by the performance of a single CPU core.

Overall, the results show that topic-based communication using selectors is much more expensive than queue-based communication and, depending on the number of topics used, it limits the scalability of the system. We demonstrated how, by using jms2009-PS, the performance and scalability of different messaging workloads and configuration scenarios can be quantified. The high configurability of the benchmark allows to tailor the workload to the user's requirements by customizing it to resemble a given application scenario. The user can then evaluate alternative ways to implement message communication in terms of their overhead, performance and scalability.

5 Conclusions

We presented a new benchmark for publish/subscribe-based messaging systems built on top of the SPECjms2007 standard workload. We discussed its configuration parameters showing how the workload can be customized to evaluate different aspects of publish/subscribe communication. Overall, jms2009-PS provides more than 80 new configuration parameters allowing the user to customize the workload in terms of the number of topics, the number of subscriptions, the number and type of selectors, and the message delivery modes.

We presented a case study demonstrating how using jms2009-PS, alternative ways to implement publish/subscribe communication in an example application scenario can be evaluated in terms of their overhead, performance and scalability. We defined three different scenarios with different communication patterns. The case study showed that the flexibility provided by topic-based publish-subscribe communication comes at a high price. The use of selectors in our scenario led to roughly two times higher CPU processing time per message. The most critical factor affecting the system performance however was the number of topics used for communication. Having a low number of topics provides maximum flexibility, however, it introduces a scalability bottleneck due to the synchronization delays. Especially, the scenario in which a single topic was used to implement a message bus clearly identifies the limitations of such an approach.

Overall, with jms2009-PS we provide a powerful benchmarking tool. Through its configurability it allows the user to evaluate publish/subscribe platforms for certain communication patterns using a complex real-world workload. Our next steps will be to extend the benchmark workload with new interactions and to prepare a complex case study analysing and comparing different scenarios on alternative platforms.

References

1. Hinze, A., Sachs, K., Buchmann, A.: Event-Based Applications and Enabling Technologies. In: Proceedings of the International Conference on Distributed Event-Based Systems (DEBS 2009). (2009)
2. Sun Microsystems, Inc.: Java Message Service (JMS) Specification - Ver. 1.1 (2002)
3. Happe, J., Friedrich, H., Becker, S., Reussner, R.H.: A pattern-based Performance Completion for Message-oriented Middleware. In: Proc. of the ACM WOSP. (2008)
4. Sachs, K., Kounev, S., Bacon, J., Buchmann, A.: Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. Performance Evaluation **66**(8) (Aug 2009) 410–434
5. Sachs, K., Kounev, S., Buchmann, A.: Performance Modeling of Message-Oriented Middleware - A Case Study. (2009) In review.
6. Sachs, K., Kounev, S., Appel, S., Buchmann, A.: A Performance Test Harness For Publish/Subscribe Middleware. In: SIGMETRICS/Performance 2009 Demo Competition, ACM (June 2009)
7. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The Many Faces of Publish/Subscribe. ACM Computing Surveys **35**(2) (2003) pages 114–131
8. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley (2003)
9. Sonic Software Corporation: Sonic Test Harness. http://communities.progress.com/pcom/docs/DOC-29828 (2005)
10. IBM Hursley: Performance Harness for Java Message Service. http://www.alphaworks.ibm.com/tech/perfharness (2005)
11. ActiveMQ: JMeter performance test. http://incubator.apache.org/activemq/jmeter-performance-tests.html (2006)
12. JBoss: JBoss JMS New Performance Benchmark. http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossJMSNewPerformanceBenchmark (2006)
13. Crimson Consulting Group: High-Performance JMS Messaging - A Benchmark Comparison of Sun Java System Message Queue and IBM WebSphere MQ. www.sun.com/software/products/message_queue/wp_JMSperformance.pdf (2003)
14. Krissoft Solutions: JMS Performance Comparison. http://www.fiorano.com/comp-analysis/jms_perf_report.htm (2006)
15. Sonic Software Corporation: Benchmarking E-Business Messaging Providers. White Paper (January 2004)
16. Carter, M.: JMS Performance with WebSphere MQ for Windows V6.0. http://www-1.ibm.com/support/docview.wss?rs=171& uid=swg24010028 (2005)
17. Fiorano Software Inc.: JMS Performance Comparison - Performance Comparison for Publish Subscribe Messaging. www.fiorano.com/whitepapers/fmq/jms_performance_comparison.php (2010)
18. Rindos, A., Loeb, M., Woolet, S.: A performance comparison of IBM MQseries 5.2 and Microsoft Message Queue 2.0 on Windows 2000. IBM SWG Competitive Technical Assessment, Research Triangle Park, NC (2001)
19. Maheshwari, P., Pang, M.: Benchmarking message-oriented middleware: TIB/RV versus SonicMQ. Concurrency Computat.: Pract. and Exper. **17**(12) (2005)
20. Menth, M., Henjes, R., Zepfel, C., Gehrsitz, S.: Throughput performance of popular JMS servers. SIGMETRICS Perform. Eval. Rev. **34**(1) (June 2006) 367–368
21. Kounev, S., Sachs, K.: Benchmarking and Performance Modeling of Event-Based Systems. it - Information Technology **51**(5) (2009) 262–269
22. Sachs, K., Kounev, S., Appel, S., Buchmann, A.: Benchmarking of Message-Oriented Middleware. In: Proc. of the DEBS 2009. (2009)