Throughput Performance of Java Messaging Services Using Sun Java System Message Queue

Robert Henjes, Michael Menth, and Christian Zepfel
University of Wuerzburg
Department of Distributed Systems
Am Hubland, 97074 Wuerzburg, Germany
E-mail: {henjes|menth|zepfel}@informatik.uni-wuerzburg.de

***Index Terms*— Brokering Middleware, Benchmarking and Assessment**

***Abstract*— The Java messaging service (JMS) is a means to organize communication among distributed applications according to the publish/subscribe principle. If the subscribers install filter rules on the JMS server, JMS can be used as a message routing platform, but it is not clear whether its message throughput is sufficiently high to support large-scale systems. In this paper, we investigate the capacity of the high performance JMS server implementation SunOneMQ by Sun Microsystems. In contrast to other studies, we focus on the message throughput in the presence of filters and show that filtering reduces the performance significantly. We also present a model that describes the service time for a single message depending on the number of installed filters and validate it by measurements. This model helps to forecast the system throughput for specific application scenarios.**

I. Introduction

The Java messaging service (JMS) is a communication middleware for distributed software components. It is an elegant solution to make large software projects feasible and future-proof by a unified communication interface which is defined by the JMS API provided by Sun Microsystems [1]. Hence, a salient feature of JMS is that applications do not need to know their communication partners, they only agree on the message format. Information providers publish messages to the JMS server and information consumers subscribe to certain message types at the JMS server to receive a certain subset of these messages. This is known as the publish/subscribe principle.

When messages must be reliably delivered only to subscribers that are presently online, the JMS in the non-durable and persistent mode is an attractive solution for the backbone of a large scale real-time communication applications. For example, some user devices may provide presence information to the JMS. Other users can subscribe to certain message types, e.g., the presence information of their friends' devices. For such a scenario, a high performance routing platform needs filter capabilities and a high capacity to be scalable for many users. In particular, the throughput capacity of the JMS server should not suffer from a large number of clients or filters.

This work was funded by Siemens AG, Munich. The authors alone are responsible for the content of the paper.

In this paper we investigate the throughput performance of the SunOneMQ JMS server implementation [2] by measurement under various conditions. In particular, we consider different numbers of publishers, subscribers, and filters, different message sizes, different kinds of filters, and filters of different complexity. Finally, we propose a mathematical model which approximates our measurement results. It is useful for the prediction of the server throughput in practice, which depends strongly on the specific application scenario.

The paper is organized as follows. In Section II we present JMS basics, that are important for our study, and consider related work. In Section III we explain our test environment and measurement methodology. Section IV shows our measurement results and proposes an analytical performance model for the JMS server throughput. Finally, we summarize our work in Section V and give an outlook on further research.

II. Background

In this section we describe the Java messaging service (JMS) and discuss related work.

A. The Java Messaging Service

Messaging facilitates the communication between remote software components. The Java Messaging Service (JMS) standardizes this message exchange. The so-called publishers generate and send messages to the JMS server, the so-called subscribers consume these messages – or a subset thereof – from the JMS server, and the JMS server acts as a relay node [3], which controls the message flow by various message filtering options. This is depicted in Figure 1. Publishers and subscribers rely on the JMS API and the JMS server decouples them by acting as an isolating element. As a consequence, publishers and subscribers do not need to know each other. The JMS offers several modes. In the persistent mode, messages are delivered reliably and in order. In the durable mode, messages are also forwarded to subscribers that are currently not connected while in the non-durable mode, messages are forwarded only to subscribers who are presently online. Thus, the server requires a significant amount of buffer space to store messages in the durable mode and it achieves a larger throughput in the non-durable mode. In this study, we only consider the persistent but non-durable mode. Information providers with similar themes may be grouped

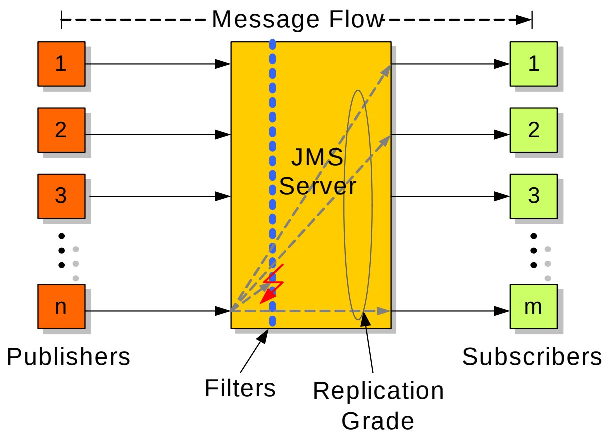


Fig. 1. The JMS server decouples publishers and subscribers.

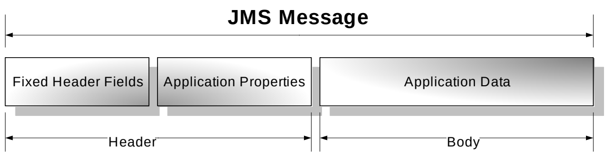


Fig. 2. JMS message structure.

together and publish to a so-called common topic; only those subscribers having subscribed for that specific topic receive their messages.

Thus, topics virtually separate the JMS server into several logical sub-servers. Topics provide only a very coarse and static method for message selection. In addition, topics need to be configured on the JMS server before system start. Filters are another option for message selection. A subscriber may install a message filter on the JMS server, which effects that only the messages matching the filter rules are forwarded instead of all messages in the corresponding topic. Each subscriber has only a single filter. In contrast to topics, filters are installed dynamically during the operation of the server. A JMS message consists of three parts that are illustrated in Figure 2: the message header, a user defined property header section, and the message payload itself [1]. So-called correlation IDs are ordinary 128 byte strings that can be set in the header of JMS messages. Correlation ID filters try to match these IDs whereby wildcard filtering is possible, e.g., in the form of ranges like [#7;#13]. Several application-specific properties may be set in the property section of the JMS message. Application property filters try to match these properties. A combination of different properties may be specified by an AND-filter which leads to more complex filters with a finer granularity compared to correlation ID filters. After all, topics, correlation ID filtering, and application property filtering are three different possibilities for message selection with different semantic granularity and different computational effort.

B. *Related Work*

The JMS is a wide-spread and frequently used middleware technology. Therefore, its throughput performance is of general interest. Several papers address this aspect already but from a different viewpoint and in different depth.

The throughput performance of four different JMS servers is compared in [4]: FioranoMQ [5], SonicMQ [6], TibcoEMS [7], and WebsphereMQ [8]. The study focuses on several message modes, e.g., durable, persistent, etc., but it does not consider filtering, which is the main objective in our work. The authors of [9] conduct a benchmark comparison for the Sun OneMQ [2] and IBM WebsphereMQ. They tested throughput performance in various message modes and, in particular, with different acknowledgement options for the persistent message mode. They also examined simple filters but they did not conduct parametric studies, and no performance model was developed. The objective of our work is the development of such a performance model to forecast the maximum message throughput for given application scenarios. A proposal for designing a “Benchmark Suite for Distributed Publish/Subscribe Systems” is presented in [10] but without measurement results. The setup of our experiments is in line with these recommendations. General benchmark guidelines were suggested in [11] which apply both to JMS systems and databases. However, scalability issues are not considered, which is the intention of our work. A mathematical model for a general publish-subscribe scenario in the durable mode with focus on message diffusion without filters is presented in [12] but without validation by measurements. In our work a mathematical model is presented for the throughput performance in the non-durable mode including filters and this model is validated by measurements. Several studies address implementation aspects of filters. A JMS server checks for each message whether some of its filters match. If some of the filters are identical or similar, some of that work may be saved by intelligent optimizations. This is discussed, e.g., in [13]. We conduct measurements for the SunOneMQ with identical and different filters in Section IV-G and the results show an increased throughput for identical filters compared to different filters.

III. Test Environment

Our objective is the assessment of the message throughput of the SunOneMQ JMS server in different application scenarios by measurements. For comparability and reproducibility reasons we describe our testbed and our measurement methodology in detail.

A. *Testbed*

Our test environment consists of five computers that are illustrated in Figure 3. Four of them are production machines and one is used for control purposes, e.g., controlling jobs like setting up test scenarios and starting measurement runs. The four production machines have a 1 Gbit/s network interface which is connected to one exclusive Gigabit switch. They are equipped with 3.2 GHz single CPUs and 1024 MB system memory. Their operating system is SuSe Linux 9.1 in standard configuration. To run the JMS environment we installed Java SDK 1.4.0, also in default configuration. The control machine is connected over a 100 Mbit/s interface to the Gigabit switch.

We installed the Sun Java System Message Queue 3 2005Q1 Platform edition (Version 3.6) [2], which is shipped with a

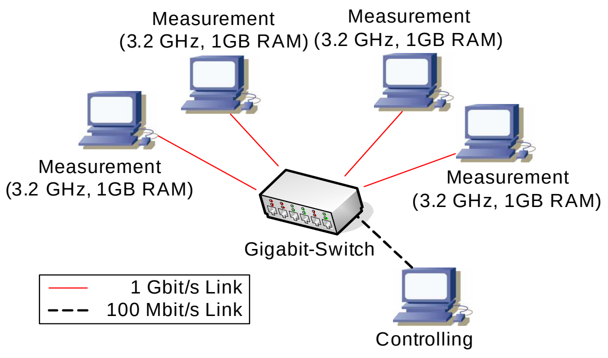


Fig. 3. Testbed environment.

trial license including all features of the enterprise edition. We use its default configuration except for the following modifications. To enable the publish/subscribe mode we set up a customized default topic. Normally, a large buffer is reserved for incoming messages. However, it is too large for our experiments, so we limit it to a maximum of 10000 messages and switch on the flow control to avoid message loss at the incoming buffer. We increased the maximum threshold for simultaneously connected publishers from 100 to 400.

We implemented test clients such that each publisher or subscriber is realized as a single Java thread, which has an exclusive connection to the JMS server component. A management thread collects the measured values from each thread and appends these data to a file in periodic intervals. In our experiments one machine is used as a dedicated JMS server, the publishers run on one or two exclusive publisher machines, and the subscribers run on one or two exclusive subscriber machines depending on the experiment. If two publisher or subscriber machines are used, the publisher or subscriber threads are distributed equally between them.

B. *Measurement Methodology*

Our objective is to measure the capacity of the JMS server. Therefore, we load it in all our experiments closely to 100% CPU load and verify that no other bottlenecks like system memory or network capacity exist on the server machine, i.e., that they have a utilization of at most 75%. The publisher and subscriber machines must not be bottlenecks, either, and they must not run at a CPU load larger than 75%. To monitor these side conditions, we use the Linux tool “sar”, which is part of the “sysstat” package [14]. We monitor the CPU utilization, I/O, memory, and network utilization for each measurement run. Without a running server, the CPU utilization of the JMS server machine does not exceed 2%, and a fully loaded server must have a CPU utilization of at least 96%.

Experiments are conducted as follows. The publishers run in a saturated mode, i.e., they send messages as fast as possible to the JMS server. However, they are slowed down if the server is overloaded because publisher side message queuing is used. To save system processing resources during the measurement phase, all JMS messages that will be ever sent by the publisher are created in advance when the publisher test clients are started. For the same reason, all connections are established before measurements are taken. Each experiment takes 100 s but we cut off the first and last 5 s due to possible warmup and cooldown effects. We count the overall number of sent messages at the publishers and the overall number of received messages by the subscribers within the remaining 90 s interval to calculate the server’s rate of received and dispatched messages. For verification purposes we repeat the measurements several times but their results hardly differ such that confidence intervals are very narrow even for a few runs.

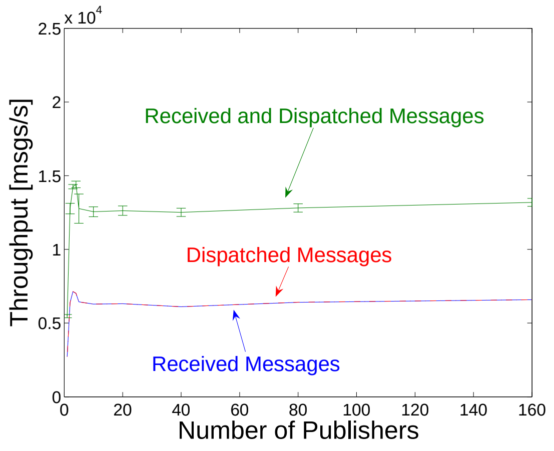


Fig. 4. Impact of the number of publishers on the message throughput in the persistent mode.

IV. Measurement Results

In this section we investigate the maximum throughput of the SunMQ JMS server. The objective is to assess and characterize the impact of specific application scenarios on its performance. In particular, we consider filters since they are essential for the use of a JMS server as a general message routing platform.

A. *Impact of the Number of Publishers*

In our first experiment, we study the impact of the number of publishers on the message throughput. Two machines carry a varying number of publishers and one machine hosts a single subscriber.

Figure 4 shows the message throughput at the JMS server in the persistent mode, i.e., lost messages are retransmitted by the JMS server and messages are preliminarily written on a disk for recovery purposes. The throughput of received and dispatched messages is plotted separately, as well as their sum which we call the overall throughput. The overall message throughput reaches its maximum rate at 14000 msgs/s for three publishers and stays then almost constantly at 13000 msgs/s for more publishers. Hence, the number of publishers hardly influences the JMS server throughput.

To assess the impact of the persistent mode, we conduct the same experiment in the non-persistent mode and the results are collected in Figure 5. The overall throughput is about 19000 msgs/s for the non-persistent mode in contrast to about 13000 msgs/s for the persistent mode. Thus, the server capacity has increased by 50%.

We also observe on our monitoring tool that the CPU utilization of the server machine is only 90% if we install

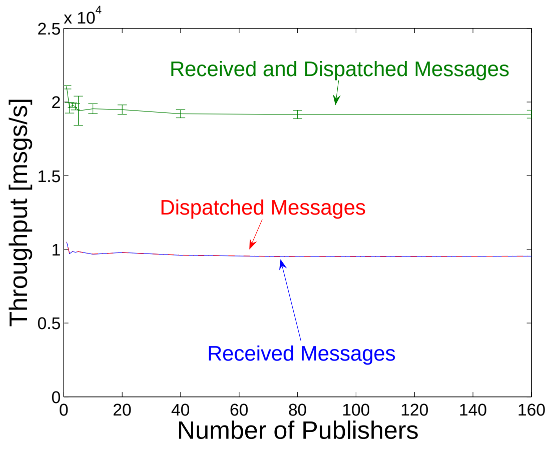


Fig. 5. Impact of the number of publishers on the message throughput in the non-persistent mode.

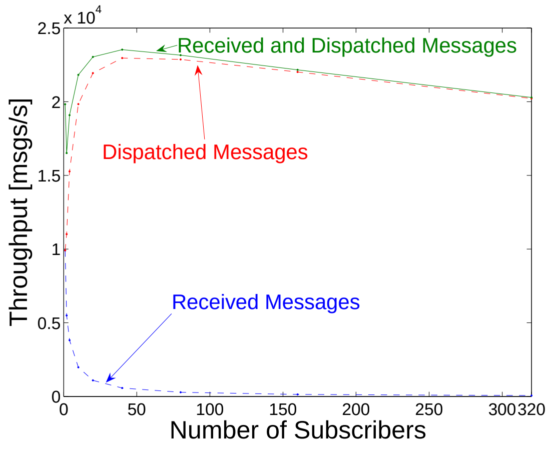


Fig. 6. Impact of the number of subscribers on the message throughput.

just one publisher thread per publisher machine. For two or more publishers, the CPU utilization is about 96% and cannot be further increased. Thus, at least two publishers are needed to fully load the JMS server. As a consequence, we use in the following experiments at least five or more publishers. We repeated the experiment three times and calculated the 99.99% confidence intervals on this basis. They are shown in the figure for the overall throughput. Obviously, they are very narrow which results from hardly varying results. Therefore, we omit them in the following figures.

B. *Impact of the Number of Subscribers*

Similarly to the above, we investigate the impact of the number of subscribers on the JMS server throughput. To that end, we have 5 publishers threads running on one machine and vary the number of subscribers on two other machines. Figure 6 shows the received, dispatched, and the overall message throughput. The overall throughput of the JMS server increases with an increasing number of subscribers to a value of about 23000 msgs/s for 40 subscribers and decreases then only slightly. This rate is significantly larger than in Figure 4 since only one subscriber is activated there.

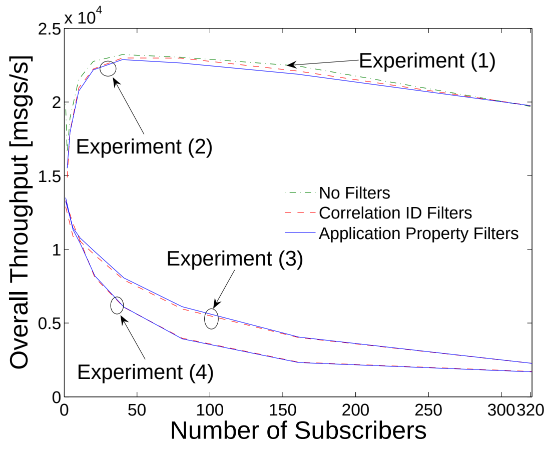


Fig. 7. Impact of filter activation and the number of subscribers on the message throughput.

Unlike in Figure 4, the received message rate decreases significantly with an increasing number of subscribers n. This can be explained as follows. No filters are applied and all messages are delivered to any subscriber. Thus, each message is replicated n times and we call this a replication grade of $r = n$. This requires more CPU cycles for dispatching messages and increases the overall processing time of a single message. As a consequence, the received message rate is reduced because the overall throughput capacity of the server stays constant. Hence, the replication grade must be considered when performance measures from different experiments are compared.

C. *Impact of Filter Activation*

We evaluate the impact of filter activation on the message throughput. We perform 4 different experiment series where all publishers send all messages with an application property or correlation ID value set to #0. We set up a variable number of n subscribers with the following filter configurations, both for application property and for correlation ID filters.

(1) No filters are installed.
(2) A filter for #0 is installed by each subscriber.
(3) A filter for #0 is installed by one subscriber and the others install a filter for #1.
(4) The subscribers install n different filters for #0, ..., #$(n-1)$

We use 5 publisher threads on a single publisher machine and a varying number of n subscriber threads on two subscriber machines.

Figure 7 illustrates the overall throughput for the above described experiments. In all experiments, correlation ID and application property filters lead to the same throughput. The curves for experiment (1) and (2) show the same high throughput, which is due to the same replication grade of $r = n$. However, if we change the replication grade to $r = 1$ in (3) by turning all filters but one to #1 instead to #0, we see a clear reduction of the throughput. Hence, the message replication grade r has a significant impact on the overall throughput. A comparison of the results for experiment (3) and (4) concludes that different filters lead to a significantly larger throughput reduction than equal filters. Obviously, the

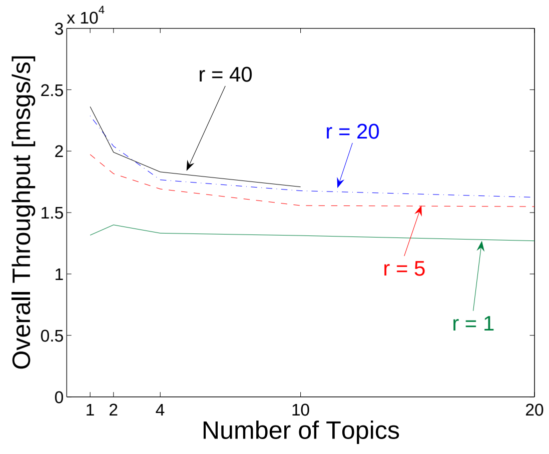


Fig. 8. Impact of the number of topics on the message throughput for different replication grades.

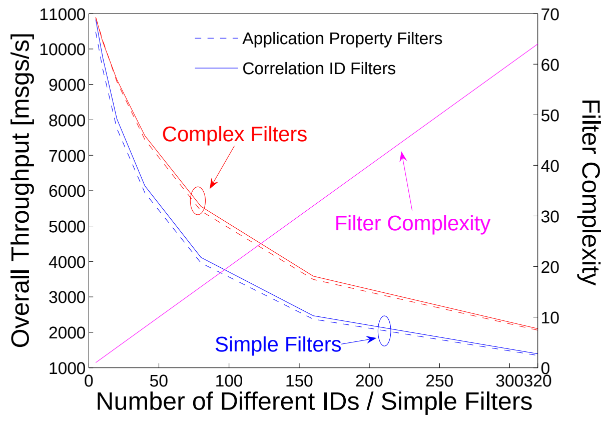


Fig. 9. Impact of simple filters and complex OR-filters on the message throughput for a replication grade of $r = 1$.

Sun Java System Message Queue implements optimized filter matching as the JMS server can take advantage of equal filters to save processing power per message. More on filter optimization can be found in [13].

D. *Impact of Topics*

Messages published to a specific topic are only dispatched to consumers who have subscribed to this particular topic. Thus, topics allow a very coarse form of message selection. In this section, we evaluate the impact of the number of topics on the message throughput for different replication grades. In our next experiment, 5 publisher threads are installed on one publisher machine and two machines host the subscribers. We vary the number of topics on the JMS server. Each publisher is connected to every topic and sends messages to them in a round robin manner. A replication grade r is obtained by registering r subscribers for each topic.

Figure 8 shows that the message throughput remains constant for a replication grade of $r = 1$, independently of the number of topics. For larger replication grades, the throughput decreases for an increasing number of topics and converges to a value of around 16000 msgs/s.

E. *Impact of Complex OR-Filters*

A single client may be interested in messages with different correlation IDs or application property values. There are two different options to get these messages. The client sets up subscribers

(1) with a simple filter for each desired message type.
(2) with a single but complex OR-filter searching for all desired message types.

We assess the JMS server performance for both option. We keep the replication grade at $r = 1$. The publishers send IDs from #1 to #n in a round robin fashion.

(1) To assess simple filters, we set up for each different ID exactly one subscriber with a filter for that ID.
(2) To assess complex filters, we set up 5 different subscribers numbered from 0 to 4. Subscriber j searches for the IDs $\#(j \cdot \frac{n}{5} + i)$ with $i \in [1; \frac{n}{5}]$ using an OR-filter.

We use in this experiment one publisher machine with 5 publisher threads and one subscriber machine with a varying number of subscribers or 5 subscribers, respectively.

Figure 9 shows the message throughput and the filter complexity depending on the number of different IDs n. The diagonal line indicates the length of the complex OR-filters. Complex filters (1) lead to up to 50% more throughput than simple filters (2) when many simple filters are used per client. Thus, it is better to use complex OR-filters than to filter each component separately by an additional subscriber. This observation holds both for application property and for correlation ID filters.

F. *Impact of Complex AND-Filters*

In the application header section of a message, multiple properties, e.g. $P_1, \ldots, P_k$, can be defined. Complex AND-filters may be used to search for specific message types. In the following, we assess the JMS server throughput for complex AND-filters. Note that complex AND-filters are only applicable for application property filtering. We use one machine with 10 publisher threads and one machine with $m = 10$ subscriber threads that are numbered by $j \in [1; m]$. We design three experiment series with a different potential for optimization of filter matching. The subscribers set up the following complex AND-filters of different length n:

(1) for subscriber j: $P_1 = \#j, P_2 = \#0, \ldots, P_n = \#0$
(2) if n is odd:
for subscriber j: $P_1 = \#0, \ldots, P_{\frac{n+1}{2}-1} = \#0, P_{\frac{n+1}{2}} = \#j, P_{\frac{n+1}{2}+1} = \#0, \ldots, P_n = \#0$
if n is even:
for subscriber j if $j \leq \frac{n}{2}$: $P_1 = \#0, \ldots, P_{\frac{n}{2}-1} = \#0, P_{\frac{n}{2}} = \#j, P_{\frac{n}{2}+1} = \#0, \ldots, P_n = \#0$
for subscriber j if $j > \frac{n}{2}$: $P_1 = \#0, \ldots, P_{\frac{n}{2}} = \#0, P_{\frac{n}{2}+1} = \#j, P_{\frac{n}{2}+2} = \#0, \ldots, P_n = \#0$
(3) for subscriber j: $P_1 = \#0, P_2 = \#0, \ldots, P_n = \#j$

The corresponding messages are sent by the publishers in a round robin fashion to achieve a replication grade of $r = 1$. Then, the filters can reject non-matching messages by looking at the first component in experiment (1), at the first half of the components in experiment (2), and at all n components in

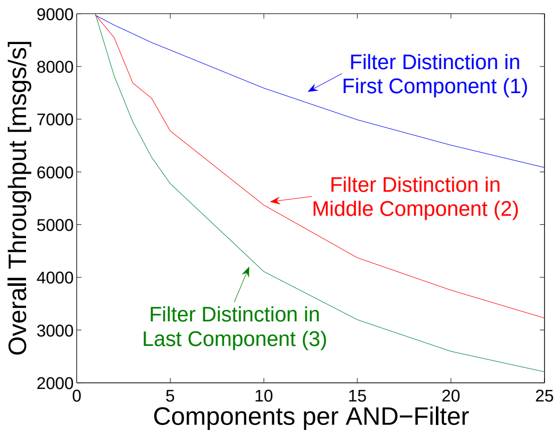


Fig. 10. Impact of an early non-match decision for AND-filters on the message throughput depending on the filter complexity for a replication grade of $r = 1$.

experiment (3). This experiment is designed such that both the replication grade and the number of subscribers is constant and that only the filter complexity n varies. To avoid any impact of different message sizes in this experiment series, we define $k = 25$ properties in all messages to get the same length.

Figure 10 shows the message throughput depending on the filter complexity n. In all scenarios, the filter complexity reduces the server capacity. Experiment (1) yields the largest message throughput, followed by experiment (2) and (3). Thus, an early reject decision of the filters shortens the processing time of a message and increases thereby the server capacity. As a consequence, practitioners should care for the order of individual components within AND-filters: components with the least match probability should be checked first.

G. *An Analytical Model for the Message Throughput*

We know from Section IV-C that different filters decrease the server capacity significantly while equal filters have hardly any impact on its performance. Furthermore, the server capacity benefits from a large message replication grade. We set up experiment series to study their joint impact on the message throughput. We propose a simple analytical model to describe these dependencies and fit the model parameters to the measurement data. The good accordance of the measured and analytical results validate our model assumption.

1) Joint Impact of the Number of Different and Equal Filters and the Replication Grade: We design such an experiment series that we can study the impact of the replication grade r, the number of different filters n_{fltr}^{diff}, and the number of all filters n_{fltr}^{all} on the message throughput. The publishers send only messages with ID #0. To achieve a replication grade of r, we set up r subscribers with a filter for ID #0. Furthermore, we install n_{diff}^{add} other different filters for values from #1 to #n_{diff}^{add}. We set up these additional filters f_r times and call f_r the filter replication factor in this experiment. We use the following values for our experiments $r \in \{1, 2, 5, 10, 20, 40\}$, $n_{diff}^{add} \in \{1, 2, 5, 10, 20, 40, 80, 160\}$, and $f_r \in \{1, 2, 4, 8\}$, and conduct them with 5 publisher threads on one publisher machine and with a variable number of $r + (n_{diff}^{add} \cdot f_r)$ subscribers on one subscriber machine.

Figures 11(a)–11(d) and Figures 12(a)–12(d) show the received and overall message throughput for the above described experiments. The server capacity clearly decreases for an increasing number of different filters n_{diff}^{add}. An increasing message replication grade r reduces the received message rate, but it increases the overall message rate. The four related figures differ by a different filter replication grade f_r, but they look very similar at the first spot. The impact of the number of all filters $n_{fltr}^{all} = r + f_r \cdot n_{diff}^{add}$ is clearly visible when we compare the right margins of the figures since the number of all filters only differs significantly if the number of additional different filters n_{diff}^{add} is large. Thereby we observe that equal filters also reduce the throughput even though they do not match.

2) A Simple Model for the Message Processing Time: We assume that the processing time of the JMS server for a message consists of four components. For each received message, there is

- a fixed time overhead t_{rcv} independently of installed filters.
- a fixed time overhead $n_{fltr}^{all} \cdot t_{fltr}^{all}$ caused by the JMS server due to the number of all installed filters. This value depends on the application scenario.
- a fixed time overhead $n_{fltr}^{diff} \cdot t_{fltr}^{diff}$ caused by the JMS server while checking which different filters are matching. This value depends on the application scenario.
- a variable time overhead $r \cdot t_{tx}$ depending on the message replication grade r. It takes into account the time the server takes to forward r copies of the message.

This leads to the following message processing time B.

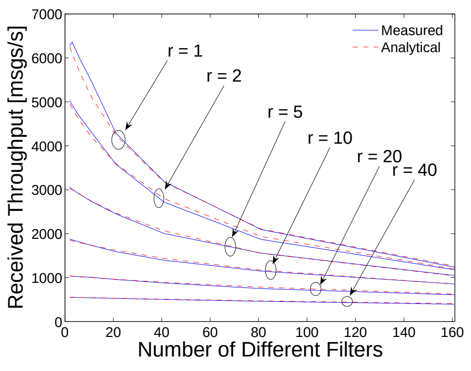
$$B = t_{rcv} + n_{fltr}^{all} \cdot t_{fltr}^{all} + n_{fltr}^{diff} \cdot t_{fltr}^{diff} + r \cdot t_{tx}. \qquad (1)$$

3) Validation of the Model by Measurement Data: The results in Figures 12(a)–12(d) show the overall throughput regarding received and sent messages. Within time B, one message is received and r messages are sent on average. Therefore, the overall throughput is given by $\frac{r+1}{B}$ and corresponds to the measurement results in Figures 12(a)–12(d). The parameters n_{fltr}^{diff}, n_{fltr}^{all}, and r for the message processing time B are known from the respective experiments. We fit the parameters t_{rcv}, t_{fltr}^{all}, t_{fltr}^{diff}, and t_{tx} by a least squares approximation [15] to adapt the model in Equation (1) to the measurement results. The results are compiled in Table I for correlation ID and application property filters. We calculate

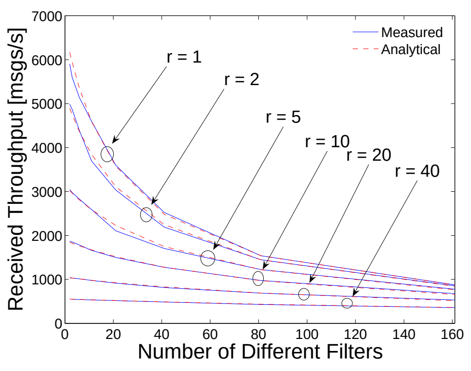
TABLE I
EMPIRICAL VALUES FOR THE MODEL PARAMETERS IN EQUATION (1).

parameters	$t_{rcv}[s]$	$t_{fltr}^{all}[s]$	$t_{fltr}^{diff}[s]$	$t_{tx}[s]$
values	$1.140 \cdot 10^{-4}$	$2.103 \cdot 10^{-6}$	$2.116 \cdot 10^{-6}$	$4.012 \cdot 10^{-5}$

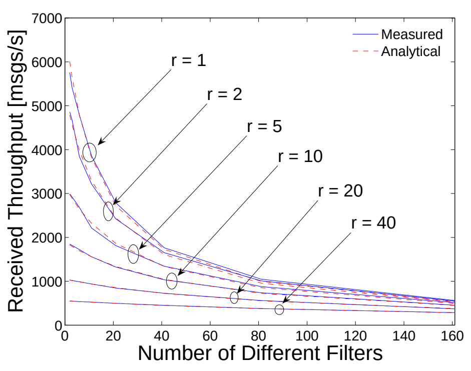
the message throughput based on these values and Equation (1) for all measured data points, and plot the results with dashed lines in Figures 11(a)–11(d) and Figures 12(a)–12(d).



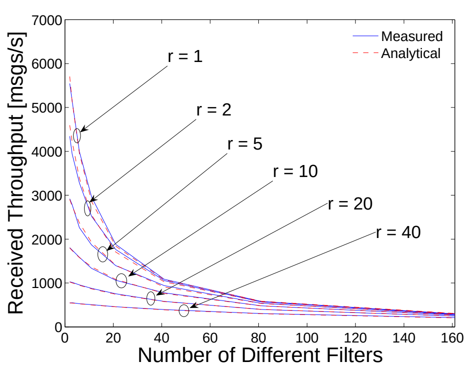
(a) Filter replication grade $f_r = 1$.



(b) Filter replication grade $f_r = 2$.

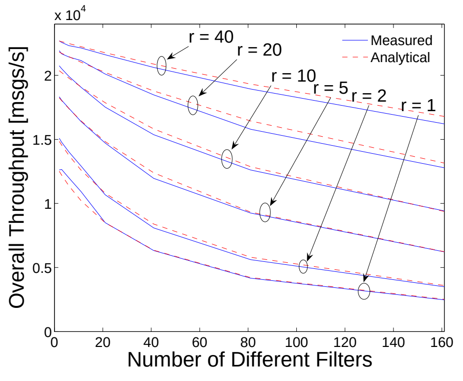


(c) Filter replication grade $f_r = 4$.

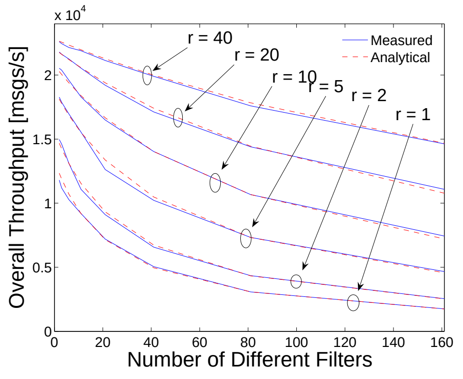


(d) Filter replication grade $f_r = 8$.

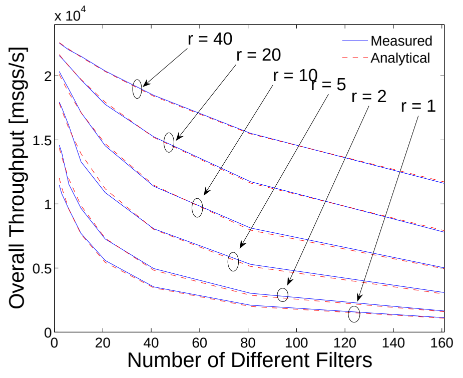
Fig. 11. Impact of the number of different filters n_{fltr}^{diff} and the message replication grade r on the received message throughput for different numbers of additional equal filters – measurements and analytical data.



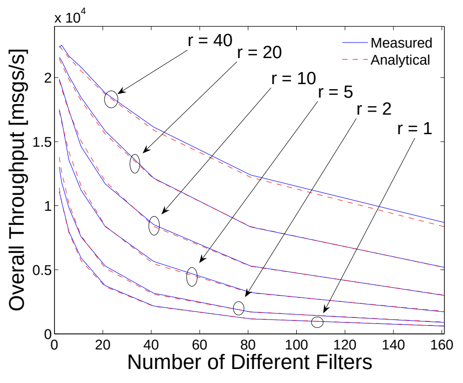
(a) Filter replication grade $f_r = 1$.



(b) Filter replication grade $f_r = 2$.



(c) Filter replication grade $f_r = 4$.



(d) Filter replication grade $f_r = 8$.

Fig. 12. Impact of the number of different filters n_{fltr}^{diff} and the message replication grade r on the overall message throughput for different numbers of additional equal filters – measurements and analytical data.

The throughput from our analytical model agrees very well with our measurements for all numbers of additional different filters n_{diff}^{add}, all replication grades r, and all filter replication grades f_r. Thus, if we know the the number of installed different filters n_{fltr}^{diff} on the JMS server, the overall number of installed filters n_{fltr}^{all}, and the mean r of the message replication grade in a certain application scenario, we have a model that predicts the average message processing time and thereby the server capacity. Thus, the model is helpful for system dimensioning.

V. Conclusion

In this work, we have investigated the capacity of the Java Messaging System (JMS) server "Sun Java System Message Queue" regarding the maximum message throughput under various conditions. We first gave a short introduction into JMS and reviewed related work. We presented the testbed and explained our measurement methodology before we conducted the experiments.

We studied the overall message throughput depending on the number of publishers and subscribers and found out that the maximum server performance can be achieved only if sufficiently many publishers and subscribers participate in the system. We showed that the message replication grade, the overall number of filters, and, in particular, the number of different filters have a large impact on the server capacity. In contrast, application property and correlation ID filters, lead to the same message throughput and also the number of topics on the server affects it overall capacity only to a minor degree. Complex OR-filters lead to a significantly higher throughput than several simple filters. For complex AND-filters, the order of the filter components has a strong impact on the server capacity. Finally, we studied the joint impact of the overall number of installed filters, the number of different filters, and the replication grade of the messages. We presented measurements for various experiments and developed and validated an analytical model for the server capacity that includes these critical parameters. This model is useful to predict the server capacity in practical application scenarios.

Currently, we investigate the message throughput of other JMS servers than the SunMQ to compare their capacity. We study the message waiting time taking into account the variability of the replication grade in practice. In addition, we intend to increase the JMS throughput, e.g., by the use of server clusters.

Acknowledgements

The authors would like to thank Dr. Andre Bondi, Prof. Christian Kanzow, and Prof. Phuoc Tran-Gia for their valuable contributions.

References

[1] *Java Message Service API Rev. 1.1*, Sun Microsystems, Inc., April 2002, http://java.sun.com/products/jms/.

[2] *Sun ONE Message Queue, Reference Documentation*, Sun Microsystems, Inc., 2005, http://developers.sun.com/prodtech/msgqueue/reference/docs/index.html.

[3] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," in *ACM Computing Surveys*, 2003.

[4] Krissoft Solutions, "JMS Performance Comparison," Tech. Rep., 2004, http://www.fiorano.com/comp-analysis/jms_perf_comp.htm.

[5] *FioranoMQ^{TM}: Meeting the Needs of Technology and Business*, Fiorano Software, Inc., Feb. 2004, http://www.fiorano.com/whitepapers/whitepapers_fmq.pdf.

[6] *Enterprise-Grade Messaging*, Sonic Software, Inc., 2004, http://www.sonicsoftware.com/products/docs/sonicmq.pdf.

[7] *TIBCO Enterprise Message Service*, Tibco Software, Inc., 2004, http://www.tibco.com/resources/software/enterprise_backbone/message_service.pdf.

[8] *IBM WebSphere MQ 6.0*, IBM Corporation, 2005, http://www-306.ibm.com/software/integration/wmq/v60/.

[9] Crimson Consulting Group, "High-Performance JMS Messaging," Tech. Rep., 2003, http://www.sun.com/software/products/message_queue/wp_JMSperformance.pdf.

[10] A. Carzaniga and A. L. Wolf, "A Benchmark Suite for Distributed Publish/Subscribe Systems," Software Engineering Research Laboratory, Department of Computer Science, University of Colorado, Boulder, Colorado, Tech. Rep., 2002.

[11] T. Wolf, "Benchmark für EJB-Transaction und Message-Services," Master's thesis, Universität Oldenburg, 2002.

[12] R. Baldoni, M. Contenti, S. T. Piergiovanni, and A. Virgillito, "Modelling Publish/Subscribe Communication Systems: Towards a Formal Approach," in 8th *International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003)*, 2003, pp. 304–311.

[13] G. Mühl, L. Fiege, and A. Buchmann, "Filter Similarities in Content-Based Publish/Subscribe Systems," *Conference on Architecture of Computing Systems (ARCS)*, 2002.

[14] *Sysstat Monitoring Utilities*, http://perso.wanadoo.fr/sebastien.godard/, Feb. 2004.

[15] C. Moler, *Numerical Computing with MATLAB*. Philadelphia, PA: Society for Industrial and Applied Mathematic (SIAM), 2004, http://www.mathworks.com/moler/chapters.html.

Robert Henjes studied computer science and linguistics at the University of Wuerzburg/Germany and received his diploma in April 2004. Mr. Henjes received an award for his diploma thesis in 2004. Currently, he is working as a researcher in the Department of Distributed Systems at the University of Wuerzburg. His research focus is performance analysis, peer-to-peer overlay structures, and publish/subscribe messaging systems.

Michael Menth studied computer science and mathematics at the Universities of Wuerzburg/Germany and Austin/Texas. He has been working as a researcher in the Department of Databases and Information Systems at the University of Ulm/Germany and he works currently again in the Department of Distributed Systems at the University of Wuerzburg. From there, he received his PhD with honors in 2004. Currently, he is assistant professor and heading the research group on next generation networks. His research focus is performance analysis and optimization of communication networks. Dr. Menth holds about 25 patent applications and has received various scientific awards for innovative work.

Christian Zepfel studied computer science and business administration at the University of Wuerzburg/Germany and received his diploma in February 2006. His research interests are grid computing, scheduling, and publish-subscribe messaging systems.