Throughput Performance of Java Messaging Services Using FioranoMQ

Robert Henjes, Michael Menth, and Sebastian Gehrsitz

Department of Distributed Systems, Institute of Computer Science
University of Würzburg, Am Hubland, D-97074 Würzburg, Germany
Phone: (+49) 931-888 6644, Fax: (+49) 931-888 6632
`{henjes,menth,gehrsitz}@informatik.uni-wuerzburg.de`

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 by Fiorano. 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.

1 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 who are presently online, the JMS in the persistent but non-durable 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 message 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 maximum throughput of the FioranoMQ JMS server implementation [2] by measurement and study its performance 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 2 we present JMS basics, that are important for our study, and consider related work. In Section 3 we explain our test environment and measurement methodology. Section 4 shows our measurement results and proposes an analytical performance model for the JMS server throughput. Finally, we summarize our work in Section 5 and give an outlook on further research.

2 Background

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

2.1 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 only forwarded to presently online subscribers. 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 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. To learn more about the different filter types, we need to have a look at the JMS message header. It consists of three parts that are illustrated in Figure 2: the message header, a user defined property header section, and the message payload itself

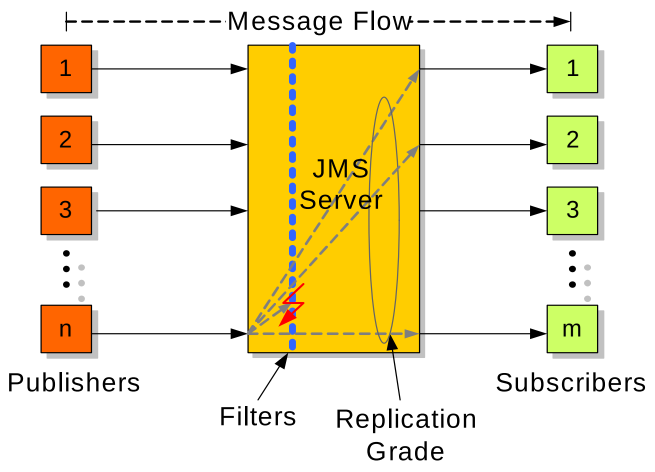


Fig. 1. The JMS server decouples publishers and subscribers.

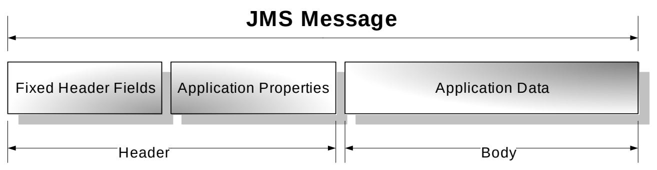


Fig. 2. JMS message structure.

[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. Unlike to correlation ID filters, a combination of different properties may be specified which leads to more complex filters with a finer granularity. 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.

2.2 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 [2], SonicMQ [5], TibcoEMS [6], and WebsphereMQ [7]. 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. Like in our investigation, this study used FioranoMQ's version 7.5. Initially, we reproduced some of the simple experiments in [4] and obtained similar results. The authors of [8] conduct a benchmark comparison for the Sun OneMQ [9] 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 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 FioranoMQ with identical and different filters in Section 4.9 and the results do not show an increased throughput for identical filters compared to different filters.

3 Test Environment

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

3.1 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 FioranoMQ version 7.5 server components as JMS server software. We used the vendor's default configuration as delivered with the test version. Our publisher and subscriber test clients are derived from Fiorano's example Java sources for measurement purposes. 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

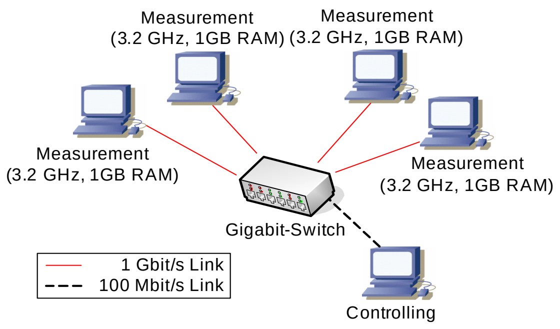


Fig. 3. Testbed environment.

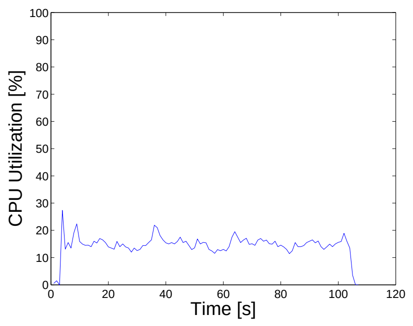
file in periodic intervals. In our experiments one machine is used as a dedicated JMS server, the publishers run on one or two exclusive publisher machines, and the subscribers run on one or two exclusive subscriber machines depending on the experiment. If two publisher or subscriber machines are used, the publisher or subscriber threads are distributed equally between them.

3.2 Measurement Methodology

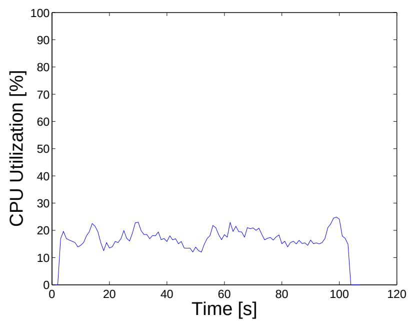
Our objective is to measure the capacity of the JMS server. Therefore, we load it in all our experiments closely to 100% CPU load and verify that no other bottlenecks like system memory or network capacity exist on the server machine, i.e., that they have a utilization of at most 75%. The publisher and 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 98%.

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.

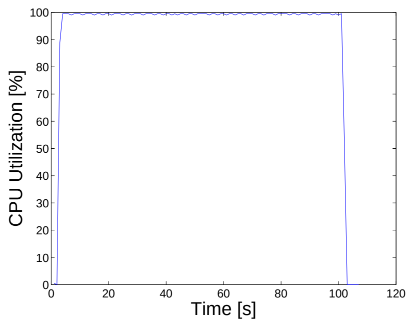
To illustrate a realization of a typical experiment, Figures 4(a)–4(c) show the utilization of the publisher, subscriber, and server machines.



(a) CPU utilization of publisher machine.



(b) CPU utilization of subscriber machine.



(c) CPU utilization of JMS server machine.

Fig. 4. Illustration of the CPU utilization of the production machines in our experiments.

4 Measurement Results

In this section we investigate the maximum throughput of the FioranoMQ 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.

4.1 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 5 shows the message throughput at the JMS server. The throughput of received and dispatched messages is plotted separately, as well as their sum which we call the overall throughput. The throughput increases with an increasing number of publishers up to 20 publishers and decreases then only slightly. Hence, the number of publishers influences the JMS server throughput to a minor extent.

We also observe on our monitoring tool that the CPU utilization of the server machine is only 72% if we install only one publisher thread per publisher machine, 97% for

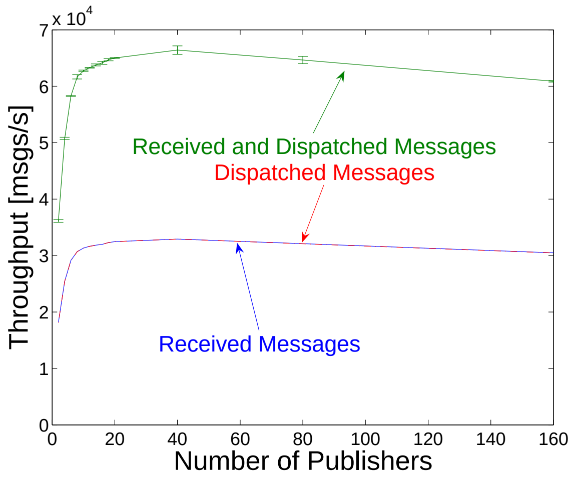


Fig. 5. Impact of the number of publishers on the message throughput.

four publishers, and 99% for six or more publishers. Thus, at least four publishers are needed to fully load the JMS server. Therefore, 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.

4.2 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 decreases only slightly with an increasing number of subscribers. Up to 320 subscribers can be connected to the JMS server simultaneously per subscriber machine, which is, however, only a restriction of our test clients.

Unlike in Figure 5, 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.

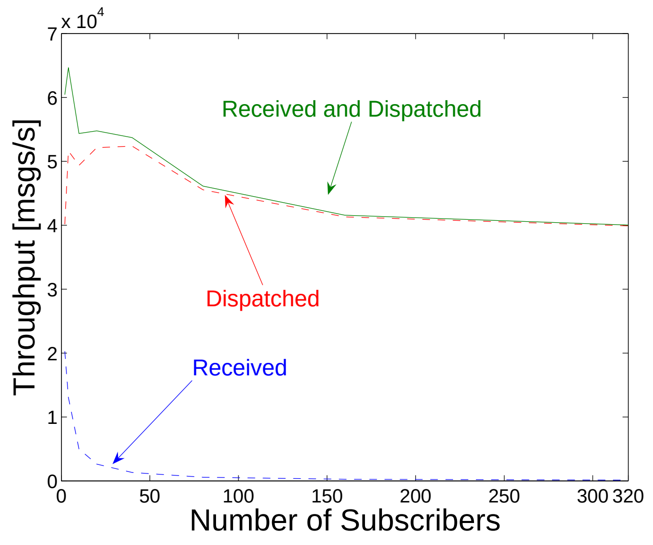


Fig. 6. Impact of the number of subscribers on the message throughput.

4.3 Impact of the Message Size

The throughput of a JMS server can be measured in messages per second (message throughput) or in transmitted data per second (data throughput). The message body size has certainly an impact on both values. We test the maximum throughput depending on the message size. We set up 10 publishers on two publisher machines and one subscriber on a single subscriber machine without any filters.

Figure 7 shows the overall throughput depending on the payload size and the corresponding message body size. The throughput in msgs/s is measured but the throughput in Mbit/s is derived from these data. The calculation of the corresponding overall message size takes into account various message headers, i.e., 40 bytes JMS header, 32 bytes TCP header, 20 bytes IP header, and 38 bytes Ethernet header, as well as TCP fragmentation. Figure 7 shows that an increasing message body size decreases the message throughput and increases the data throughput significantly. For small message bodies, the message throughput is limited by 61000 msgs/s while for very large message sizes, the data throughput is limited by about 600 Mbit/s. Obviously, the network interface of the JMS server becomes the system bottleneck. We proved this speculation by measuring the maximum throughput of a single TCP connection which amounts to at most 350 Mbit/s in one direction. In all the other experiments, the default value for the message body size is 0 bytes.

4.4 Impact of Filter Activation

We evaluate the impact of correlation ID and application property filter activation on the message throughput. We perform three different experiment series that are designed in such a way that a mean replication grade of $r = n$ achieved for the sake of a fair

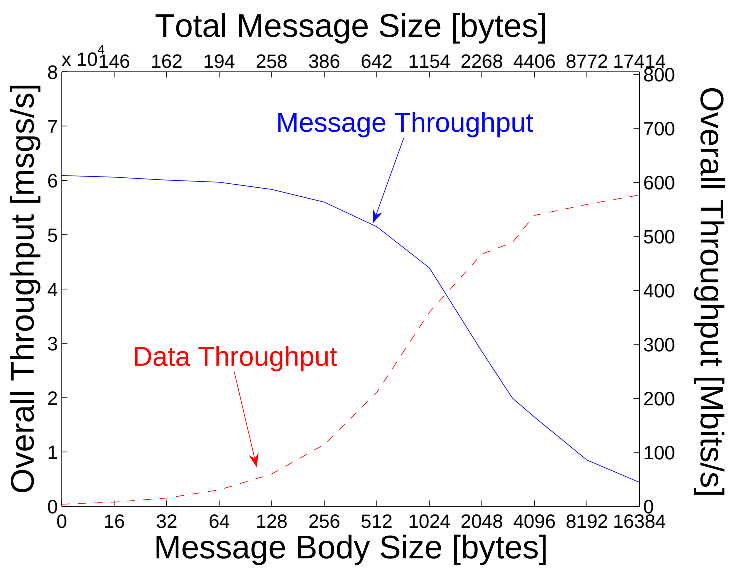


Fig. 7. Impact of the message body size on the message and data throughput.

comparison. The publishers send all messages with correlation ID or an application property value set to #0. We set up a variable number of *n* subscribers with the following filter configurations.

(1) No filters are installed.
(2) A correlation ID filter for #0 is installed by each subscriber.
(3) An application property filter for #0 is installed by each subscriber.

We use 5 publisher threads on a single publisher machine and a single machine for the subscribers in the filter experiment; for the experiment without filters we need two subscriber machines like in Section 4.2 to avoid that they are heavily loaded.

Figure 8 shows the overall throughput without filters (1), for correlation ID filters (2), and for application property ID filters (3) for an increasing number of subscribers. The throughput is maximal for two or four subscribers, respectively, and decreases slightly for an increasing number of subscribers when no filters are installed (1), but it decreases drastically when filters are activated ((2) and (3)). Application property filters lead to about half the throughput compared to correlation ID filters. Thus, filtering reduces the capacity of JMS servers significantly.

4.5 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

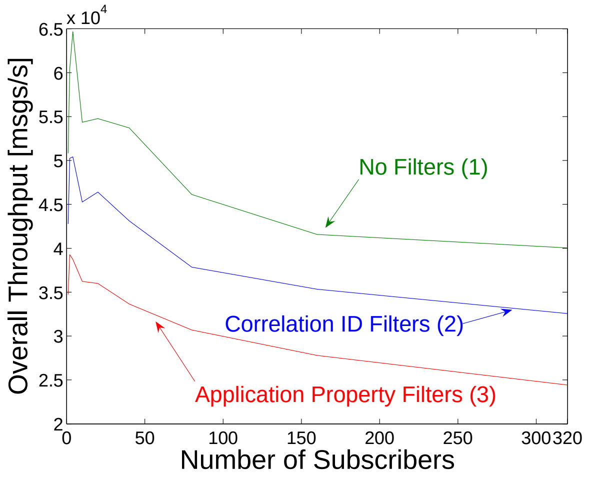


Fig. 8. Impact of filter activation and the number of subscribers on the message throughput.

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.

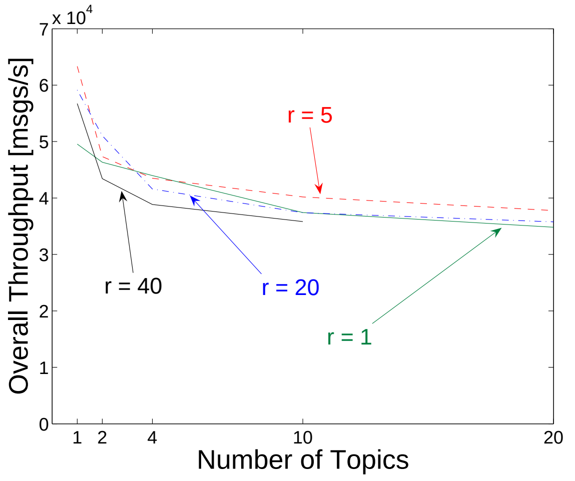


Fig. 9. Impact of the number of topics on the message throughput for different replication grades.

Figure 9 shows that the message throughput decreases slightly for an increasing number of topics and that it seems to converge to a value of around 35000 msgs/s independently of the replication grade.

4.6 Comparison: Impact of Topics and Filters

In the following experiment we compare the message throughput for topic, correlation ID, and application property filtering since all three options can be used for message selection. For the sake of a fair comparison, we design the experiments in such a way that they have all a replication grade of $r = 1$.

(1) In the case of topics, only one subscriber without filter is connected to each topic. Each publisher is connected to all topics and sends messages to them in a round robin fashion.
(2) In the case of correlation ID filters, each publisher sends correlation ID numbers from #1 to #n in a round robin fashion. Furthermore, we set up exactly n subscribers with correlation ID filters in such a way that there is exactly one matching filter installed for each sent correlation ID number.
(3) In the case of application property filters, experiment (2) is adapted by substituting correlation IDs by application property values.

We use one subscriber machine and one publisher machine with 5 publisher threads for all experiments.

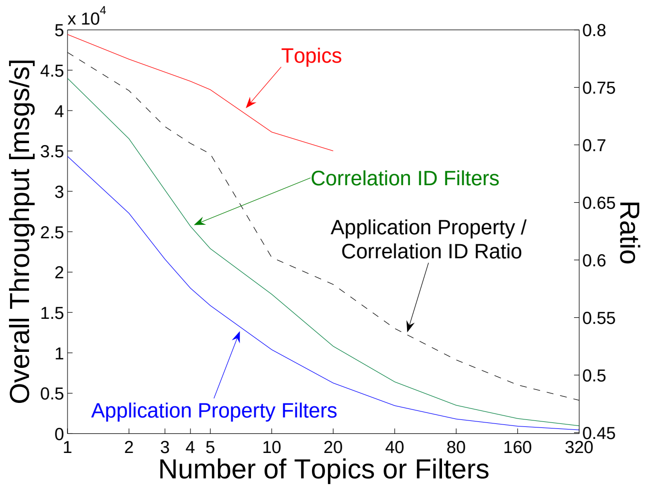


Fig. 10. Impact of the number of filters or topics on the message throughput.

Figure 10 shows the overall message throughput depending on the number of topics or filters, respectively. In each case, the throughput decreases for an increasing number of topics or filters. The throughput for topics decreases the least for an increasing number and is around 35000 msgs/s for 20 different topics. Due to the high manual configuration overhead, we limited this experiment to 20 different topics. In contrast, the throughput for filters is steadily decreased by an increasing number of filters. In addition, the impact of filters on the server performance is apparently significantly larger than the one of topics. The figure also shows the ratio of the throughput for application property and correlation ID filters. The application property filtering leads to only 50% of the JMS server capacity compared to correlation ID filtering when many filters are activated. Thus, the finer the configured message selection granularity of the JMS is, the lower is its maximum throughput. Note that the throughput curves for both filter types differ significantly from those in Figure 8. This can be explained by the different replication grade.

4.7 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 now 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.

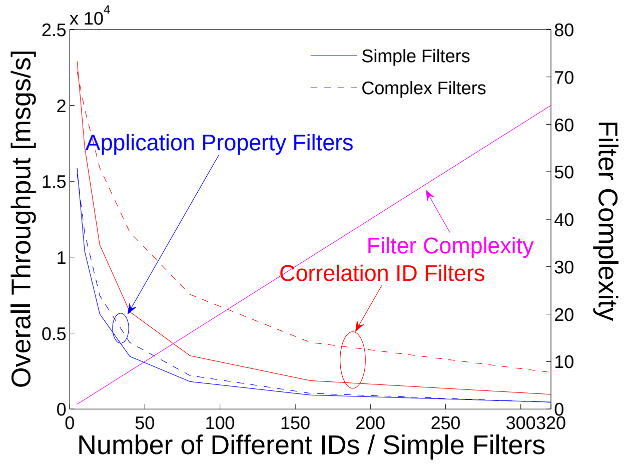


Fig. 11. Impact of simple filters and complex OR-filters on the message throughput for a replication grade of $r = 1$.

Figure 11 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. For correlation ID filters, complex filters (1) lead to about 100% more throughput than simple filters (2) when 40 or more 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. For application property filters, the absolute throughput is considerably smaller than with correlation ID filters and the use of complex filters (2) brings hardly any advantage compared to simple filters (1).

4.8 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.

In the following, 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 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 12 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.

4.9 An Analytical Model for the Message Throughput

We have learned from Section 4.4 and Section 4.6 that both the number of filters and the replication grade impact the JMS server capacity. In this section, we investigate

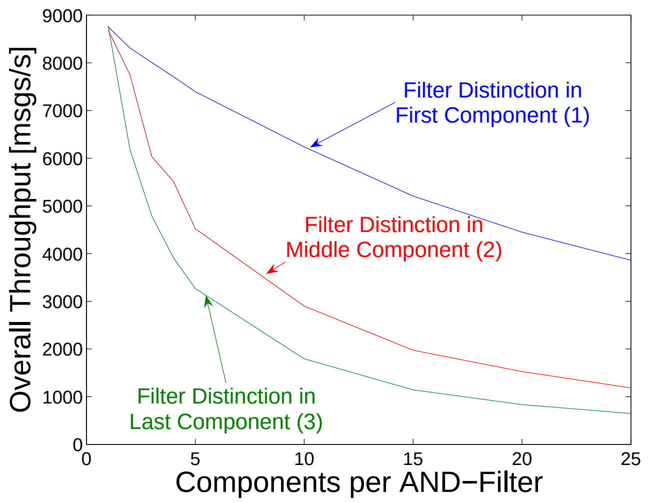


Fig. 12. Impact of an early non-match decision for AND-filters on the message throughput depending on the filter complexity for a replication grade of $r = 1$.

their joint impact and present a simple model to forecast the server performance for a given number of filters and for an expected replication grade. This model is validated by measurements.

Joint Impact of the Number of Filters and the Replication Grade We set up experiments to conduct parameter studies regarding the number of installed filters and the replication grade r of the messages. We use one publisher and one subscriber machine. Five publishers are connected to the JMS server and send messages with correlation ID #0 or application property value #0 in a saturated way. Furthermore, $n+r$ subscribers are connected to the JMS server, r of them filter for correlation ID or application property attribute #0 while the other n subscribers filter for different correlation IDs. Hence, $n+r$ filters are installed altogether. This setting yields a message replication grade of r. We choose replication grades of $r \in \{1,2,5,10,20,40\}$ and $n \in \{5,10,20,40,80,160\}$ additional subscribers.

Figure 13 shows the message throughput for correlation ID filters depending on the number of installed filters $n_{fltr} = n+r$ and on the replication grade r. The solid lines show the measured throughput. An increasing number of installed filters reduces obviously the message throughput of the system and an increasing replication grade increases the system performance to a certain extent. Similar measurements are obtained for application property filtering, which are illustrated in Figure 14. The basic performance behavior is the same, but the absolute overall message throughput is about 50% compared to the one of correlation ID filters when many filters are applied. We get the same results for both experiments if all the n non-matching filters search for the same value, e.g. for #1, and if they look for different values, e.g. for #1, ..., #n. Thus, we cannot find any throughput improvement if equal filters are used instead of different filters [13].

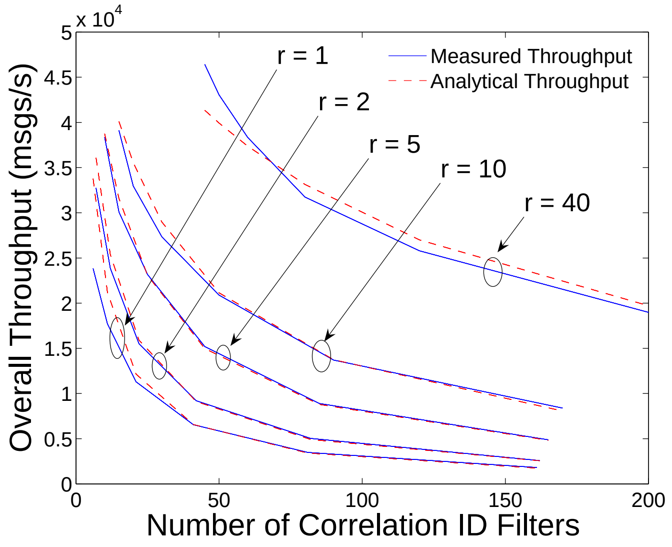


Fig. 13. Impact of the number of filters n_{fltr} and the message replication grade r on the overall message throughput in case of correlation ID filters – measurements and analytical data.

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

- a fixed basic time overhead t_{rcv} independently of filter installations.
- a fixed time overhead $n_{fltr} \cdot t_{fltr}$ 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.

$$B = t_{rcv} + n_{fltr} \cdot t_{fltr} + r \cdot t_{tx} \tag{1}$$

Validation of the Model by Measurement Data The results in Figures 13 and 14 show the overall throughput regarding received and sent messages.

Within time B, one message is received and r messages are dispatched by the server. Thus, the overall throughput is given by $\frac{r+1}{B}$ and corresponds to the measurement results in Figures 13 and 14. The parameters n_{fltr} and r for the message processing time B are known from the respective experiments. We fit the parameters t_{rcv}, t_{fltr}, 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 1 for correlation ID and application property filters. We calculate the message throughput based on these values and Equation (1) for all measured data points, and plot the results with dashed lines in Figures 13 and 14. The throughput from our analytical model agrees very well with our measurements for

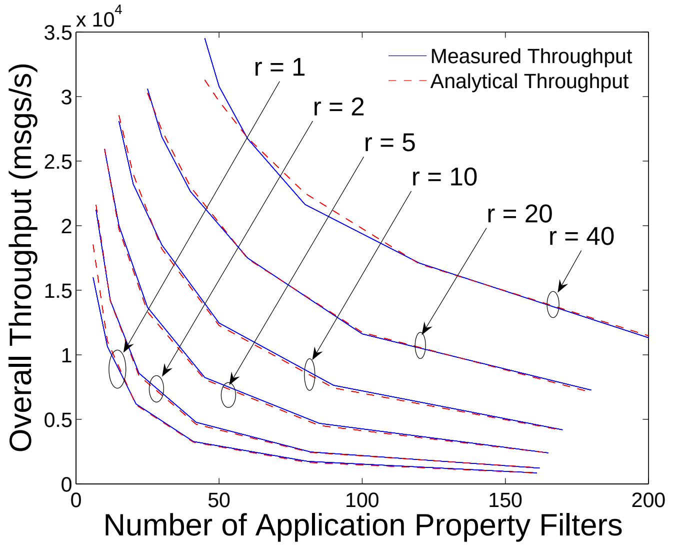


Fig. 14. Impact of the number of filters n_{fltr} and the message replication grade r on the overall message throughput in case of application property filters – measurements and analytical data.

Table 1. Overhead values for the model of the message processing time in Equation (1).

overhead type	$t_{rcv}[s]$	$t_{fltr}[s]$	$t_{tx}[s]$
corr. ID filtering	$8.52 \cdot 10^{-7}$	$7.02 \cdot 10^{-6}$	$1.70 \cdot 10^{-5}$
app. prop. filtering	$4.10 \cdot 10^{-6}$	$1.46 \cdot 10^{-5}$	$1.62 \cdot 10^{-5}$

all numbers of filters n_{fltr} and all replication grades r. Thus, if we know the the number of installed filters n_{fltr} on the JMS server and the mean r of the message replication grade in a certain application scenario, we have a model that allows prediction of the average message processing time.

4.10 Impact of the Aging of the SuSe Linux OS

We conducted an experiment over 4 weeks. The FioranoMQ server software was re-run every day. After 4 weeks the JMS server throughput was reduced by about 30% compared to the start of the experiments. We booted the computer anew and the experiments yielded again the originally higher throughput. We do not have an explanation for that phenomenon, but we want to report this as an interesting observation. To avoid a corruption of the measurement results by the aging of the system, we booted the JMS server regularly.

5 Conclusion

In this work, we have investigated the capacity of the FioranoMQ Java Messaging System (JMS) server 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 first the impact of different numbers of publishers and subscribers on the server capacity as well as the impact of the JMS message size on the achievable throughput both in terms of messages rate and bit rate. Then we considered the concepts of topics, correlation ID filters, and application property filters. They allow message selection with an increasing flexibility but this also reduces the server capacity significantly. Complex filters decrease obviously the server throughput. The use of complex OR-filters instead of several simple filters increases the server capacity if correlation ID filters are used but it yields hardly any benefit for application property filters. In case of AND-filters, the order of the components in the filter expression has a severe impact on the server capacity. The filters should be arranged in a such way that least probable components should be checked first. Finally, we studied the joint impact of the number of installed filters and the replication grade of the messages. The server capacity decreases for an increasing number of filters but it increases for an increasing message replication grade. Thus, the message throughput depends heavily on the specific application scenario. We developed an analytical model for the server capacity and validated it by our measurements. This model is useful to predict the server capacity in practical application scenarios.

Currently, we are investigating the message throughput of other JMS servers than the FioranoMQ to compare their capacity. Furthermore, we study the message waiting time taking into account the variability of the replication grade of the messages which leads to different message processing times. In addition, we intend to study how the JMS throughput can be increased, e.g., by the use of server clusters.

Acknowledgements

The authors would like to thank Prof. Tran-Gia for his fruitful discussions.

References

1. Sun Microsystems, Inc.: Java Message Service API Rev. 1.1. (2002) `http://java.sun.com/products/jms/`.
2. Fiorano Software, Inc.: FioranoMQ^{TM}: Meeting the Needs of Technology and Business. (2004) `http://www.fiorano.com/whitepapers/whitepapers_fmq.pdf`.
3. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The Many Faces of Publish/Subscribe. In: ACM Computing Surveys. (2003)
4. Krissoft Solutions: JMS Performance Comparison. Technical report (2004) `http://www.fiorano.com/comp-analysis/jms_perf_comp.htm`.
5. Sonic Software, Inc.: Enterprise-Grade Messaging. (2004) `http://www.sonicsoftware.com/products/docs/sonicmq.pdf`.
6. Tibco Software, Inc.: TIBCO Enterprise Message Service. (2004) `http://www.tibco.com/resources/software/enterprise_backbone/message_service.pdf`.
7. IBM Corporation: IBM WebSphere MQ 6.0. (2005) `http://www-306.ibm.com/software/integration/wmq/v60/`.

8. Crimson Consulting Group: High-Performance JMS Messaging. Technical report (2003) `http://www.sun.com/software/products/message_queue/wp_JMSperformance.pdf`.
9. Sun Microsystems, Inc.: Sun ONE Message Queue, Reference Documentation. (2005) `http://developers.sun.com/prodtech/msgqueue/reference/docs/index.html`.
10. Carzaniga, A., Wolf, A.L.: A Benchmark Suite for Distributed Publish/Subscribe Systems. Technical report, Software Engineering Research Laboratory, Department of Computer Science, University of Colorado, Boulder, Colorado (2002)
11. Wolf, T.: Benchmark für EJB-Transaction und Message-Services. Master's thesis, Universität Oldenburg (2002)
12. Baldoni, R., Contenti, M., Piergiovanni, S.T., Virgillito, A.: Modelling Publish/Subscribe Communication Systems: Towards a Formal Approach. In: 8^{th} International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003). (2003) 304–311
13. Mühl, G., Fiege, L., Buchmann, A.: Filter Similarities in Content-Based Publish/Subscribe Systems. Conference on Architecture of Computing Systems (ARCS) (2002)
14. `http://perso.wanadoo.fr/sebastien.godard/`: Sysstat Monitoring Utilities. (2004)
15. Moler, C.: Numerical Computing with MATLAB. Society for Industrial and Applied Mathematic (SIAM), Philadelphia, PA (2004) `http://www.mathworks.com/moler/chapters.html`.