University of Würzburg
Institute of Computer Science
Research Report Series

Throughput Comparison of Professional JMS Servers

Michael Menth, Robert Henjes,
Sebastian Gehrsitz, and Christian Zepfel

Report No. 380 March 2006

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
{menth|henjes|gehrsitz|zepfel}@informatik.uni-wuerzburg.de

Throughput Comparison of Professional JMS Servers

Michael Menth, Robert Henjes,
Sebastian Gehrsitz, and Christian Zepfel
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
{menth|henjes|gehrsitz|zepfel}@informatik.uni-wuerzburg.de

Abstract

The Java messaging service (JMS) facilitates communication among distributed software components 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 three high performance JMS server implementations: FioranoMQ, SunMQ, and WebshereMQ. 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 present models for the message processing time of each server and validate them by measurement. These models depend on the number of installed filters and the replication grade of the messages, and predict the overall message throughput for specific application scenarios. Finally, we illustrate the use of these models by comparing the message throughput of the three servers in four different 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 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

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

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.

In this paper we compare the performance of the FioranoMQ [2], the SunMQ [3], and the WebsphereMQ [4] JMS server implementation. We evaluate their maximum throughput 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. We propose a mathematical model depending on the number of filters and the message replication grade to approximate the processing time of a message for each server type. Finally, we show the usefulness of these models in practice by predicting and comparing the message throughput of the three server types in four different application scenarios.

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 measurement results of rather simple experiments whereas Section 5 proposes quite complex measurement setups to derive mathematical models for the message processing time from their results. These models are useful to predict the server throughput for specific application scenarios which is demonstrated in a comparative study in Section 6. Finally, we summarize our work in Section 7.

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 [5], 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. In this study, we consider the persistent but non-durable mode if not mentioned differently.

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

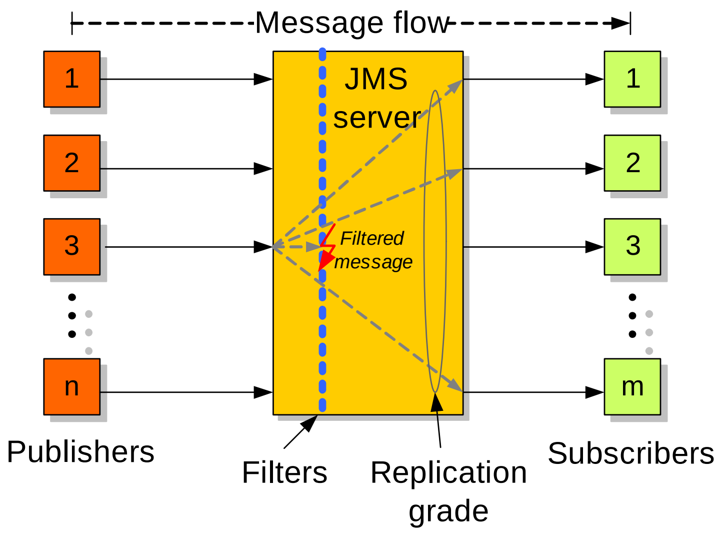


Figure 1: The JMS server decouples publishers and subscribers.

that only the messages matching the filter rules are forwarded instead of all messages in the corresponding topic. In contrast to topics, filters are installed dynamically during the operation of the server. Figure 2 shows that a JMS message consists of three parts: 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. 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.

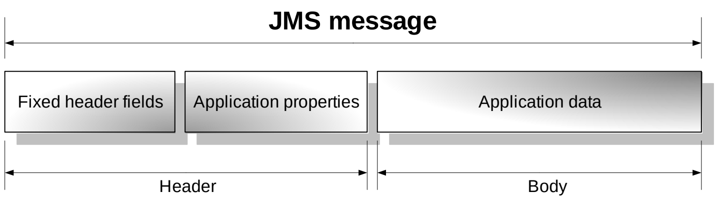


Figure 2: JMS message structure.

2.2 Related Work

A general introduction to publish/subscribe systems can be found in [6]. It presents a taxonomy of existing systems and compares qualitatively the capabilities of different concepts. The SIENA middleware is based on the publish/subscribe principle and has been presented in [7] with the objective to achieve an Internet-scale scalability for event notification services. This and several other papers [8, 9] focus on an efficient design of publish/subscribe systems to build high-performance message routing platforms. However, they do not provide measurement results or performance models for the message throughput of publish/subscribe systems. Another optimization aspect addresses semantic issues like uncertainties in queries [10] which obviously leads to a tradeoff between missed information and redundant message delivery. Several studies address implementation aspects of filters [11, 12]. A JMS server checks for each message whether some of its filters match. If some of the filters are identical or similar, filter matching can be optimized [13]. We conduct measurements for the SunMQ with identical and different filters in Section 5 and the results show an increased throughput for identical filters compared to different filters.

The JMS is a wide-spread and frequently used middleware technology. Therefore, its throughput performance is of general interest and several papers address this aspect already from a different viewpoint and in different depth. A mathematical model for publish/subscribe communication systems in the durable mode is presented in [14], but its focus lies rather on semantic than performance issues. Guidelines for benchmarking distributed publish/subscribe systems are given in [15] but without measurement results. The setup of our experiments is in line with these recommendations. On the one hand, there are only a few academic studies considering the throughput performance by experimental measurements [16, 17, 18], but they do not address the server capacity in the presence of message filters. In [16] the throughput performance of Tibco Rendezvous and SonicMQ JMS server is compared while [17] contrasts two leading JMS products whose names are not revealed. The authors of [18] show that the throughput of JMS servers suffers tremendously from durable subscriptions when mobile users hand over. On the other hand, performance studies of publish/subscribe systems are of great interest such that whitepapers compare the throughput of various commercial servers. The throughput performance of four different JMS servers is compared in [19]: FioranoMQ [2], SonicMQ [20], TibcoEMS [21], and WebsphereMQ [4]. 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 [22] conduct a benchmark comparison for the SunMQ [3] and the 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 a comparison of the throughput performance of the FioranoMQ, the SunMQ, and the WebsphereMQ JMS server in various application scenarios. In particular, we focus on the impact of filters and develop a performance model for the server capacity to predict the maximum message throughput for specific application scenarios.

3 Test Environment

Our objective is the assessment of the message throughput of the FioranoMQ, SunMQ, and WebsphereMQ JMS server by hardware measurement under various conditions. For comparability and reproducibility reasons we describe our testbed, the server installations, 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. 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 publishers or subscribers are distributed equally between them. 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.

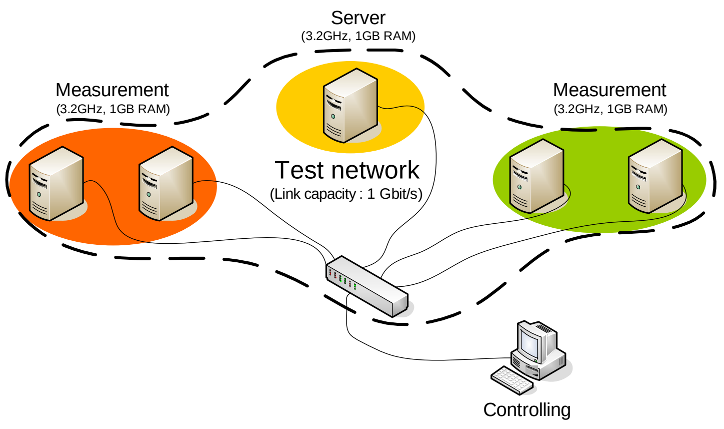


Figure 3: Testbed environment.

3.2 Server Installation

We briefly describe the installation of the three considered server types.

We installed the FioranoMQ [2] version 7.5 server components as JMS server software. We used the vendor's default configuration as delivered with the test version. We start the server in the superuser mode; otherwise user restrictions can limit the number of simultaneously connected clients to the FioranoMQ kernel.

We installed the Sun Java System Message Queue 3 2005Q1 Platform edition (Version 3.6) [3], which is shipped with a 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. Like above, we increased the maximum threshold for simultaneously connected publishers from 100 to 400.

We installed the the IBM Websphere MQ 6.0 Trial version [4] on the server machine with the default configuration except for the following modifications. For performance reasons we deactivated the security module because our experiments do not focus on security issues. We raised the internal restriction regarding the number of parallel connections to the queue manager from the default value 100 to 500. To conduct our experiments, we used WebshereMQ's integrated publish/subscribe feature instead of an additional broker.

3.3 Measurement Methodology

Our objective is the measurement of the JMS server capacity. Therefore, we load the server 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 [23]. 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. 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.

4 Measurement Results

In this section we investigate the maximum throughput of the FioranoMQ, SunMQ, and WebsphereMQ JMS servers. The objective is to assess and characterize the impact of specific application scenarios on their 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 4 shows the received 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. FioranoMQ achieves the highest received message throughput with 32000 msgs/s, followed by SunMQ with 9500 msgs/s and WebsphereMQ with 1000 msgs/s. Thus, the message throughput spans several orders of magnitude. FioranoMQ requires 40 publishers to achieve its maximum throughput whereas SunMQ and WebsphereMQ need only 5 publishers to achieve a typical throughput. As a consequence, we use in the following experiments at least 5 or more publishers.

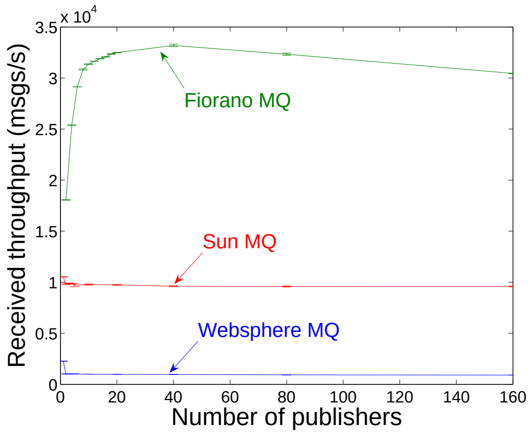


Figure 4: Impact of the number of publishers on the received message throughput in the persistent mode.

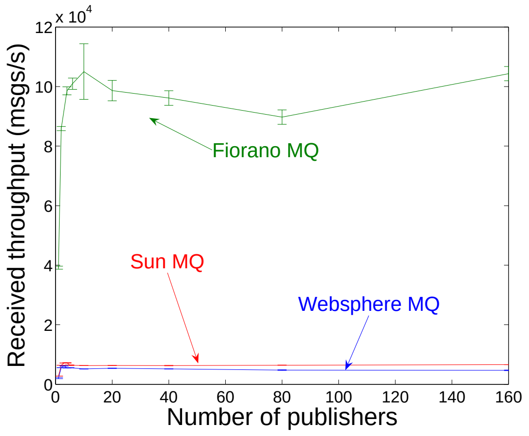


Figure 5: Impact of the number of publishers on the received message throughput in the non-persistent mode.

To assess the impact of the persistent mode, we conduct the same experiments in the non-persistent mode and the results are collected in Figure 5. The received throughput is about 100000 msgs/s for FioranoMQ, 13500 msgs/s for SunMQ, and 9500 msgs/s for WebsphereMQ. Thus, the message throughput is significantly increased, in particular for WebsphereMQ. However, especially for WebsphereMQ we observe a high packet loss rate of about 8% under full load.

We repeated both experiment series several times and calculated the 95% confidence intervals on this basis. They are shown in both figures. Obviously, they are very narrow which results from hardly varying results. Therefore, we omit them in the following figures for the sake of clarity.

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 for the SunMQ. 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.

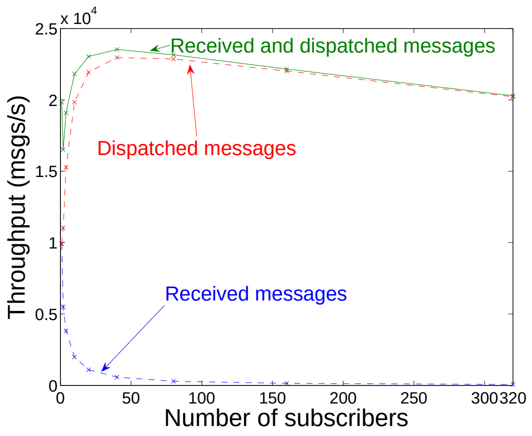


Figure 6: SunMQ: Impact of the number of subscribers on the received, dispatched, and overall message throughput.

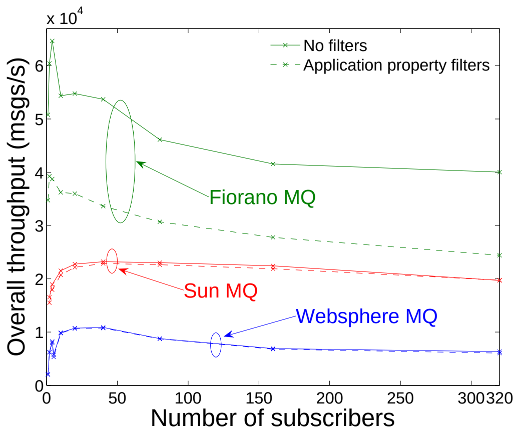


Figure 7: Impact of filter activation and the number of subscribers on the message throughput.

4.3 Impact of Filter Activation

We evaluate the impact of filter activation on the message throughput. Figure 7 shows the overall message throughput depending on the number of subscribers with and without filters. We used 5 publishers in all experiments. FioranoMQ achieves its maximum throughput for 5 subscribers, about 40000 msgs/s for many subscribers without filters, but only 25000 msgs/s with application property filters. Correlation ID filters lead to 33000 msgs/s, which is omitted in the graph for the sake of clarity. SunMQ and WebsphereMQ require both 20 or 40 subscribers to reach their maximum throughput of 23000 msgs/s or 11000 msgs/s, respectively. In contrast to FioranoMQ, they show the same capacity with and without filters. Thus, they are hardly slowed down by the filtering engine in this experiment. However, this finding is only valid if the message replication grade increases with the number of subscribers, which is a rather artificial case. In Section 5, we study the joint impact of filters and the replication grade for each server type in more detail. After all, we learn from these results that at least 5 subscribers are required for future experiments to get a representative value for the maximum overall message throughput.

4.4 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. For each server type we use such an experiment set up that the server achieves a sufficiently high throughput, i.e. 10 publishers threads on two machines send messages to the FioranoMQ and WebsphereMQ, and 5 are sufficient for the SunMQ server. We use one subscriber on a single machine for the FioranoMQ, 2 for the SunMQ, and 5 for the WebsphereMQ. Figure 8 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.

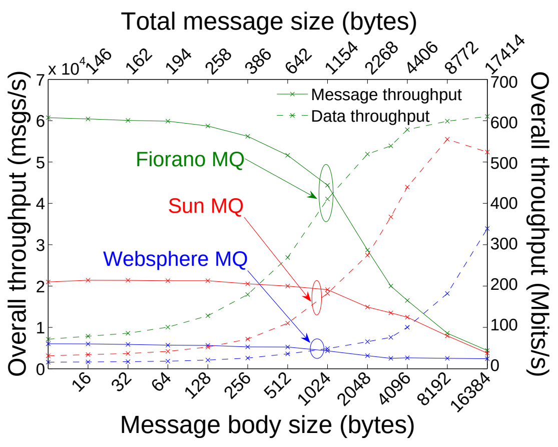


Figure 8: Impact of the message body size on the message and data throughput.

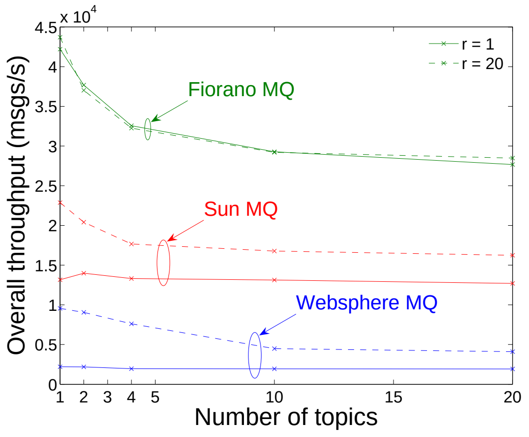


Figure 9: Impact of the number of topics on the message throughput for different replication grades.

Figure 8 shows that an increasing message body size decreases the message throughput and increases the data throughput significantly. For small message bodies of 0 bytes, the message throughput is limited by 61000 msgs/s for FioranoMQ, 21000 msgs/s for SunMQ, and 6000 msgs/s for WebsphereMQ. For large message bodies of 16384 bytes, the throughput is limited by 4400 msgs/s, 3800 msgs/s, and 2400 msgs/s. Thus, the capacity ratio between the server types changes. The performance degradation of the servers has different shapes and this shape depends also on the application scenario of the server, i.e. the number of publishers and subscribers, the message replication grade and the filters. The overall consumed bandwidth is 614 Mbit/s, 525 Mbit/s, and 336 Mbit/s for the three different server types. This is very large, but it does not yet reach the bidirectional TCP transmission limit of the network for which we measured simultaneously 350 Mbit/s each in both directions. In our experiments, the default value for the message body size is 0 bytes.

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 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 9 shows the message throughput for all 3 server types. FioranoMQ achieves the highest throughput followed by SunMQ and by WebsphereMQ. The throughput converges asymptotically to a value that is specific to the message replication grade. This value increases mostly with the replication grade. That finding holds for all server types. The limiting throughput for many topics and a replication grade larger than 1 and amounts to 28000 msgs/s for FioranoMQ, 15000 msgs/s for SunMQ, and 4000 msgs/s for WebsphereMQ. Hence, topics can be used for coarse message selection with a moderate performance loss for many topics. In particular, this impact is weaker than the one of the message replication grade.

4.6 Impact of Complex OR-Filters

A single client may be interested in messages with different 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 options. 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.

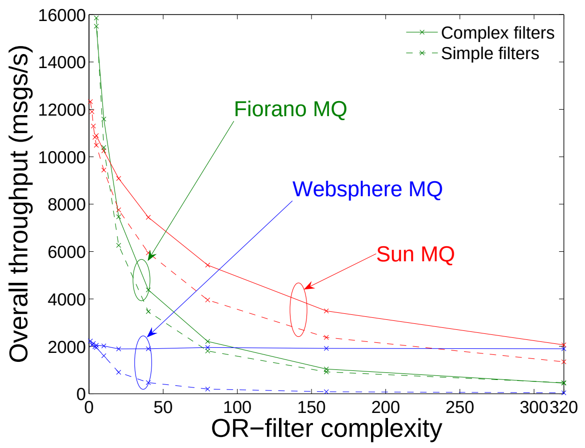


Figure 10: Impact of simple filters and complex OR-filters on the message throughput for a replication grade of $r = 1$.

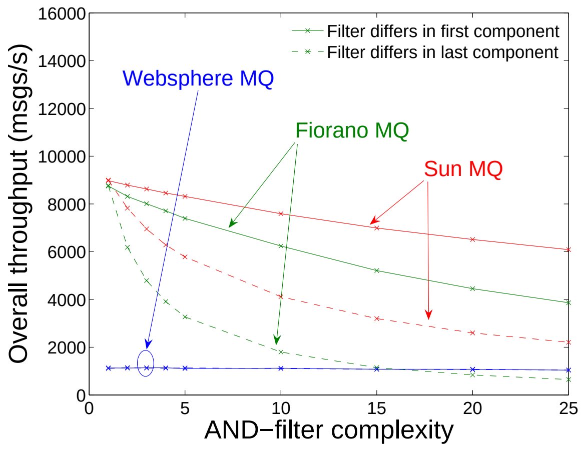


Figure 11: Impact of an early non-match decision for AND-filters on the message throughput depending on the filter complexity.

Figure 10 shows the message throughput depending on $\frac{n}{5}$, which is the number of components in the complex OR-filter complexity or the number of different simple subscribers per client. Firstly, we observe that the message throughput decreases significantly for an increasing number of installed simple filters. This is unlike in Figure 7 and the difference comes from the smaller replication grade which is $r = 1$ instead of $r = n$. Thus, the number of filters decreases the message throughput considerably if the messages are not forwarded to all subscribers, which is usually intended to avoid with filters. Secondly, we observe that complex filters (2) lead to a larger throughput than simple filters (1) but the extent of the performance gain depends strongly on the server type. For FioranoMQ, complex filters lead to a slightly larger throughput than multiple simple filters per client. For SunMQ, complex filters yield a performance gain of roughly 100%, and for WebsphereMQ, complex filters even avoid the performance loss that is observed for simple filters. Thus, the handling of simple and complex filters by WebsphereMQ takes the same computation effort. However, this finding holds certainly only to a certain extent.

4.7 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 filters but not for correlation ID filters. 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 two experiment series with different potential for optimization of filter matching. The subscribers set up the following complex AND-filters of different length n:

(1) for subscriber j: $P_1 = \#j, P_2 = \#0, ..., P_n = \#0$

(2) for subscriber j: $P_1 = \#0, P_2 = \#0, ..., P_n = \#j$

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 already reject non-matching messages by looking at the first filter component (1) or only by looking at all n filter components (2). The experiments are 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 11 shows the message throughput depending on the filter complexity n. The filter complexity reduces the server capacity significantly for FioranoMQ and SunMQ. Experiment (1) yields a considerably larger message throughput than experiment (2). 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. For WebsphereMQ, the message throughput is neither affected by the filter complexity nor by the position of the component which is decisive for the rejection of a message. As a consequence, we assume that the filter logic of WebsphereMQ has a relatively high general filter overhead without optimization for complex AND-filters since simple filter expressions take the same filtering effort as complex filter expressions.

5 Performance Models for the Joint Impact of the Number of Filters and the Replication Grade

We know from Section 4.3 and Section 4.6 that both the number of filters and the replication grade influence the capacity of JMS servers. In this section, we investigate their joint impact on the message throughput for each server type in detail and provide mathematical approximation models. To that end, we design first experiments with a varying number of filters and a varying replication grade. We take measurements, suggest mathematical models that are able to capture the gained throughput curves, and fit the model parameters by a least squares approximation. The measured and the analytical throughput agree very well for all three server types such that the performance models can be used to predict the server capacity for specific application scenarios.

5.1 Performance Model for FioranoMQ

We describe the experiment series for the FioranoMQ, suggest a suitable mathematical approximation model for the server throughput, and fit the corresponding model parameters.

5.1.1 Experiment Setup and Measurement Results

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 application property value #0 while the other n subscribers filter for value #1. 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.

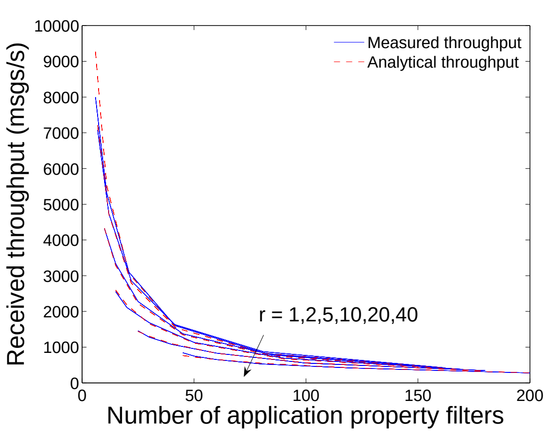


Figure 12: FioranoMQ: Impact of the number of filters n_{fltr} and the message replication grade r on the overall message throughput – measurements and analytical data.

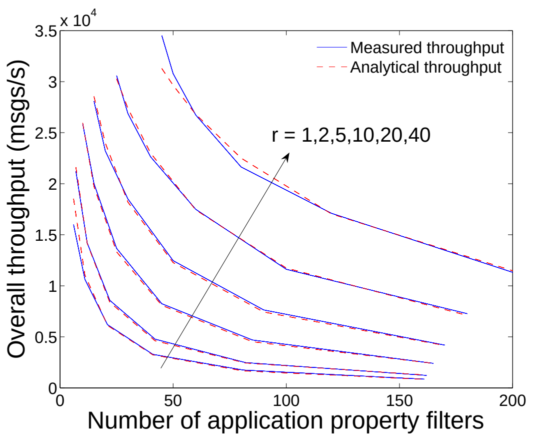


Figure 13: FioranoMQ: Impact of the number of filters n_{fltr} and the message replication grade r on the received message throughput – measurements and analytical data.

Figures 12 and 13 show the received and overall message throughput for application property 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 server. An increasing replication grade decreases the received message throughput, but it increases overall message throughput of the server to a certain extent. We obtain similar measurement curves with about 100% more throughput for correlation ID filters. In addition, we conduct the same experiment series with the n non-matching filters set to #1, ..., #n. They lead to the exactly same results as in Figures 12 and 13. Thus, we cannot find any throughput improvement if equal filters are used instead of different filters.

5.1.2 Performance Model for the Message Processing Time

The message processing time is the inverse of the received message throughput. Figure 12 shows that it depends both on the number of filters n_{fltr} and the replication grade r. Therefore, we propose a very simple model for the message processing time B:

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

The parameter t_{rcv} is a fixed time overhead for each received message. The filtering effort increases linearly with the number of filters n_{fltr} and the time to check a single filter is t_{fltr}. Finally, t_{tx} describes the time to dispatch and to send a single message for a matching filter.

5.1.3 Validation of the Model by Measurement Data

The results in Figures 12 and 13 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 Figure 13. The parameters n_{fltr} and r for the message processing time in Equation (1) are known from the respective experiments. We fit the parameters t_{rcv}, t_{fltr}, and t_{tx} by a least squares approximation to adapt the model in Equation (1) to the measurement results. The resulting parameter values are compiled in Table 1 for correlation ID and application property filters.

Table 1: Empirical values for the model parameters of the message processing time in Equation (1).

parameter	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	$4.10 \cdot 10^{-6}$	$1.46 \cdot 10^{-5}$	$1.62 \cdot 10^{-5}$

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 12 and 13. The throughput from our analytical model agrees very well with our measurements for all numbers of filters n_{fltr} and all replication grades r.

5.2 Performance Model for SunMQ

We describe the experiment series for the SunMQ, suggest a suitable mathematical approximation model for the server throughput, and fit the corresponding model parameters. Note that the model for SunMQ is more complex than the model for FioranoMQ.

5.2.1 Experiment Setup and Measurement Results

We performed the same experiment like above for the SunMQ and found out that it matters whether non-matching filters are equal or different. Thus, 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 value #0. To achieve a replication grade of r, we set up r subscribers with a filter for value #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.

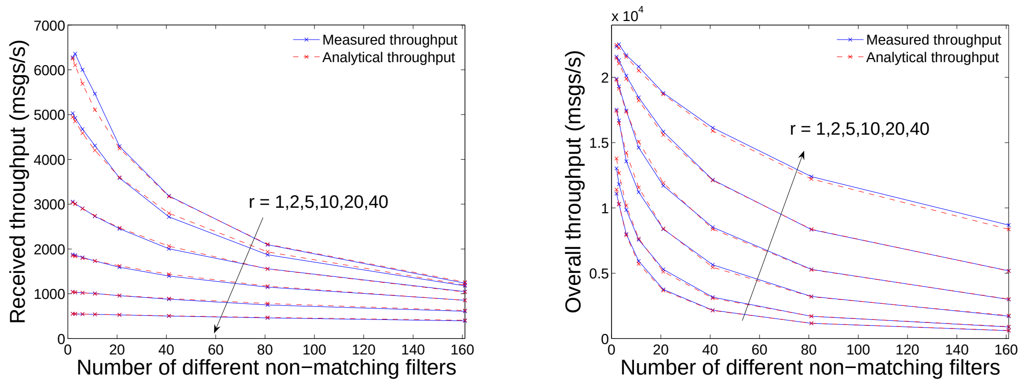
Figure 5.2.2 shows the received and overall message throughput for this experiment series. The server capacity clearly decreases for an increasing number of different filters n_{diff}^{add}. An increasing message replication grade r reduces the received message rate, but it increases the overall message 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.

5.2.2 A Simple Model for the Message Processing Time

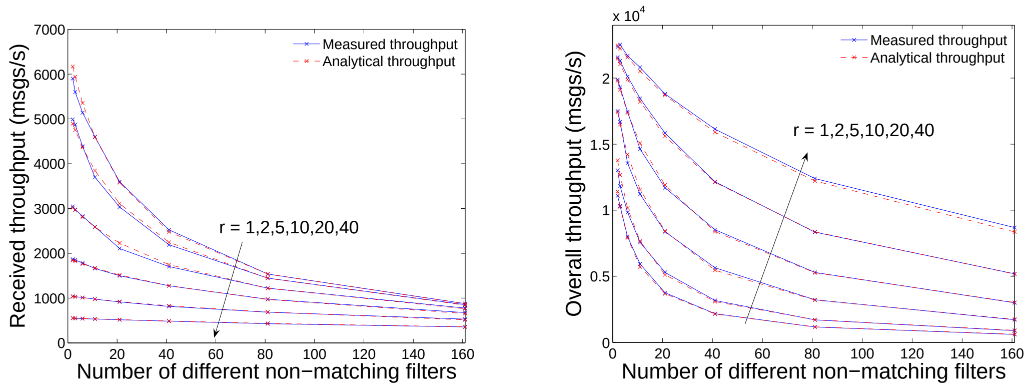
The message processing time is the inverse of the received message throughput. The Figures 14(a)–14(d) on the left show that it depends on the number of additional filters n_{fltr}^{add}, the filter replication factor f_r, and the replication grade r. We propose a simple model for the message processing time B that relies on $n_{fltr}^{all} = r + f_r \cdot n_{diff}^{add}$ and $n_{fltr}^{diff} = n_{diff}^{add} + 1$:

$$B \quad = \quad t_{rcv} + n_{fltr}^{all} \cdot t_{fltr}^{all} + n_{fltr}^{diff} \cdot t_{fltr}^{diff} + r \cdot t_{tx}. \qquad (2)$$

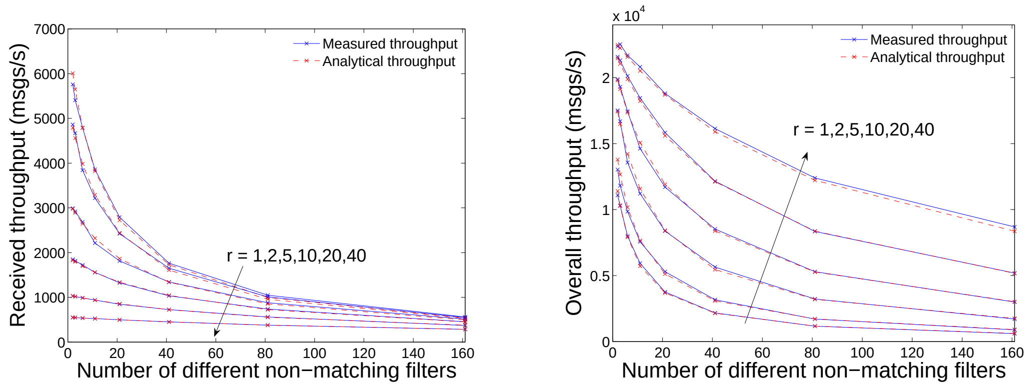
The parameter t_{rcv} is a fixed time overhead for each received message. The filtering effort increases linearly with the number of all filters n_{fltr}^{all} and the time to check a single filter is t_{fltr}^{all}. Different filters impose an extra overhead of $n_{fltr}^{diff} \cdot t_{fltr}^{diff}$. Finally, t_{tx} describes the time to dispatch and to send a single message for a matching filter.



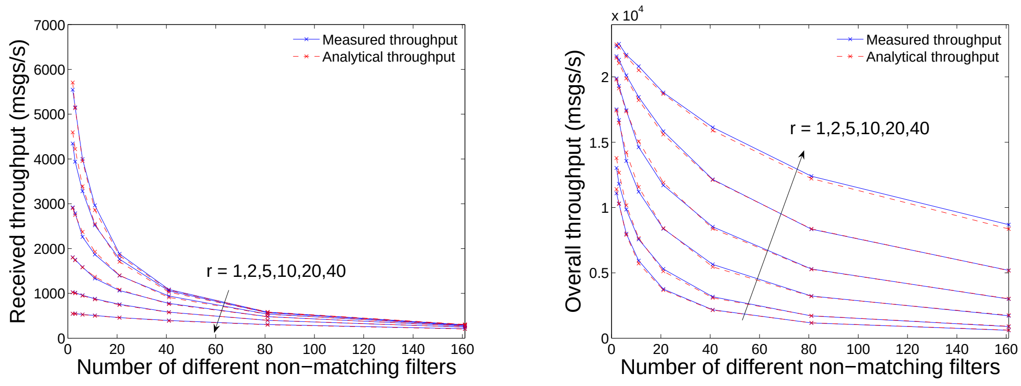
(a) Filter replication grade $f_r = 1$.



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



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



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

Figure 14: SunMQ: Impact of the number of different filters n_{fltr}^{diff} and the message replication grade r on the received and overall message throughput for different numbers of additional equal filters – measurements and analytical data.

5.2.3 Validation of the Model by Measurement Data

The results in the right column of the Figures 14(a)–14(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 14(a)–14(d) on the right column. 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 to adapt the model in Equation (2) to the measurement results. The results are compiled in Table 2 for correlation ID and application property filters. We calculate the message throughput based on these values

Table 2: Empirical values for the model parameters of the message processing time in Equation (2).

parameter	t_{rcv} (s)	t_{fltr}^{all} (s)	t_{fltr}^{diff} (s)	t_{tx} (s)
value	$1.118 \cdot 10^{-4}$	$2.200 \cdot 10^{-6}$	$1.785 \cdot 10^{-6}$	$4.008 \cdot 10^{-5}$

and Equation (2) for all measured data points, and plot the results with dashed lines in Figure 5.2.2. The throughput from our analytical model agrees very well with our measurement results.

5.3 Performance Model for WebsphereMQ

We conduct an experiment series to study the joint impact of the replication grade r and the number of installed filters n_{fltr} for the WebsphereMQ, suggest a suitable mathematical approximation model for the server throughput, and fit the corresponding model parameters. Note that WebshereMQ requires a substantially different model for the message processing time compared to FioranoMQ and SunNQ.

5.3.1 Experiment Setup and Measurement Results

We set up the same series of experiments like for the FioranoMQ in Section 5.1.1. Figure 15 shows the received message throughput 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 filters reduces the received message throughput of the system which is obviously independent of the replication grade. This is different to the results for FioranoMQ and SunMQ in Section 5.1.1 and Section 5.2.1. Figure 16 shows the resulting overall message throughput. It decreases also with an increasing number of filters, but it rises with the replication grade. We have performed the same experiments for correlation ID filters, too, and obtained the same measurement results. Thus, correlation ID and application property filters lead to the same throughput both for SunMQ and WebsphereMQ.

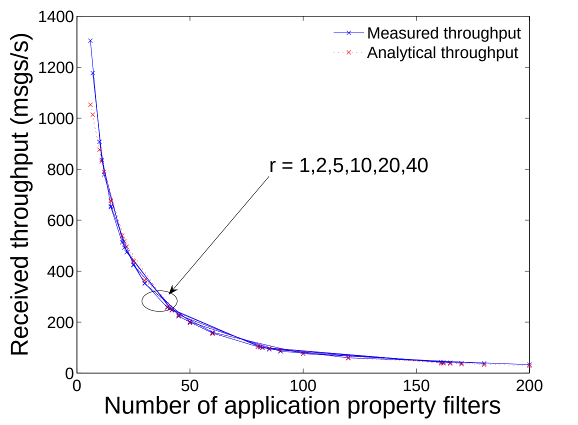


Figure 15: WebsphereMQ: Impact of the number of filters n_{fltr} and the message replication grade r on the received message throughput – measurements and analytical data.

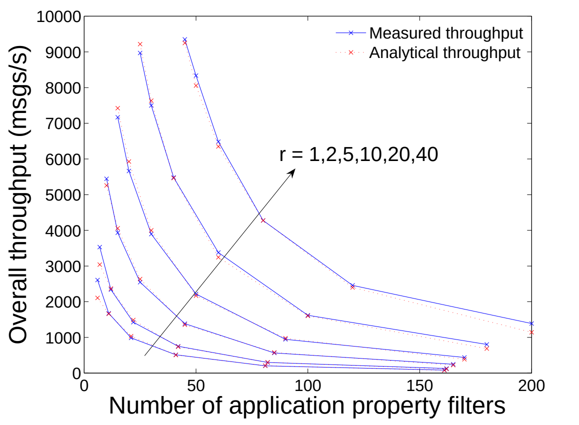


Figure 16: WebsphereMQ: Impact of the number of filters n_{fltr} and the message replication grade r on the overall message throughput – measurements and analytical data.

5.3.2 A Simple Model for the Message Processing Time

Figure 15 shows that the message processing time depends only on the number of filters n_{fltr}. In contrast to FioranoMQ and SunMQ, it does not depend on the replication grade r. Thus, the time to send messages is obviously so small that it is not noticeable for a replication grade of up to $r = 40$. A linear model like for the FioranoMQ or the SunMQ in Section 5.1.2 and Section 5.2.2 does not work for the approximation of the above measurement results. Therefore, we propose the following model for the message processing time B:

$$B \quad = \quad t_{rcv} + n_{fltr} \cdot \sqrt{(n_{fltr})} \cdot t_{fltr}. \qquad (3)$$

The parameter t_{rcv} is a fixed time overhead for each received message. The filtering effort affects the processing time with a supplement of $n_{fltr} \cdot \sqrt{(n_{fltr})} \cdot t_{fltr}$. Hence, it increases more than linearly with the number of installed filters n_{fltr}.

5.3.3 Validation of the Model by Measurement Data

As mentioned above, the received and the overall throughput can be analytically calculated by $\frac{1}{B}$ and $\frac{r+1}{B}$. Again, we adapt the model parameters t_{rcv} and t_{fltr} in Equation (3) by a least squares approximation and obtain for them the values $t_{rcv} = 7.03 \cdot 10^{-4}$ and $t_{fltr} = 1.1017 \cdot 10^{-5}$. We calculated the received and overall throughput for all measured data points based on these values and Equation (3), and plot them with dashed lines in Figures 15 and 16. The throughput from our analytical model agrees very well with our measurement data for all numbers of filters n_{fltr} and all replication grades r. For a very high message replication grade like $r = 80$, the prediction tends to be incorrect since the time to send the 80 outgoing messages imposes additional time which is not captured by the model. However, the model predicts the overall message throughput of the server quite accurately for a wide range of realistic parameters n_{fltr} and r.

5.4 Summary of the Performance Models

We have investigated the joint impact of the number of filters and the replication grade on the server capacity of FioranoMQ, SunMQ, and WebsphereMQ. FioranoMQ leads to enhanced throughput for correlation ID filters compared to application property filters while the filter type does not lead to different results for SunMQ and WebshpereMQ. Only SunMQ implements an optimized filter matching algorithm such that equal filters can be handled more efficiently than different filters. The message replication grade has an impact on the message processing time for FioranoMQ and SunMQ, but not for WebsphereMQ as long as a replication grade of $r = 40$ is not exceeded. The filtering effort for SunMQ and FioranoMQ increases at most linearly with the number of installed filters whereas WebshpereMQ showed a worse filter scalability in our experiments. As a consequence, the models we developed for the message processing time of each server type were substantially different. They are useful to predict the server capacity for specific application scenarios. Thus, they can be used to dimension the number of servers in a network. The throughput comparison of the three different server platforms helps in general to decide which of these solutions satisfies the requirements of a special distributed application from a performance point of view.

6 Application Example

We assume a distributed notification service, i.e., producers generate so-called events and consumers are notified about them. A JMS server can be used to implement such a service. We assume many producers and 100 consumers. There are many event types, but each consumer is interested in only one. The consumers may use filters with $n_{fltr}^{all} = 100$ to get only the relevant events; otherwise, they are notified about all events and have to process a higher load. The consumers are interested in $n_{fltr}^{diff} \in \{1, 10, 100\}$ different events. We predict the JMS server throughput based on the results of our study, in particular for different message replication grades r. Large replication grades occur if several clients filter for the same events. If no filters are used, we consult Figure 7 to determine the received throughput. If filters are applied, we use Equation (1), Equation (2), and Equation (3) with the respecitve parameters for application property filtering to calculate the server capacity. We have compiled the throughput of received messages at the servers in Table 3.

The use of filters increases the throughput performance in these application scenarios for FioranoMQ and for SunMQ but not for WebsphereMQ. However, the use of filters is not only recommended to increase the server throughput but also to protect the consumers from undesired load if they are only interested in 1% or 10% of the messages. We immediately realize that FioranoMQ and SunMQ are superior to WebsphereMQ in all considered application scenarios. Therefore, we discuss only the performance of these two solutions. Without filters, FioranoMQ has twice the capacity of SunMQ and each consumers receives all messages. With

Table 3: Throughput capacity of the FioranoMQ, SunMQ, and WebsphereMQ JMS server for different application scenarios with 100 subscribers and an overall number of $n_{fltr}^{all} = 100$ filters if filters are used.

n_{fltr}^{diff} if applicable	repl. grade r	Fiorano capacity (msgs/s)	Sun capacity (msgs/s)	Websphere capacity (msgs/s)
no filters	100	456	228	90
100	1	676	1817	85
10	1	676	2566	85
1	1	676	2676	85
10	10	615	1333	85

all consumers having a filter installed, the throughput increases to 676 msgs/s for FioranoMQ, and for SunMQ to 1817, 2566, or 2677 msgs/s if the number of different filters n_{fltr}^{diff} is 100, 10, or 1. This holds for a message replication grade of $r = 1$. In this case, the clients get only 1% of all messages. For a replication grade of $r = 10$, the clients get 10% of all messages. Then, FioranoMQ achieves a throughput of 615 msgs/s and SunMQ 1333 msgs/s if $n_{fltr}^{diff} = 10$. Thus, SunMQ has twice the capacity of FioranoMQ if filters are applied.

After all, only FioranoMQ and SunMQ can be considered as high throughput performance JMS platform. FioranoMQ is the better choice without filters whereas SunMQ performs better when filters are applied. From a throughput performance point of view, WebsphereMQ is clearly inferior both to FioranoMQ and SunMQ. However, Websphere comes with a wealth of other functionality and the mere consideration of the throughput performance of its JMS module is then certainly not a sufficient criterion against this solution, in particular, if high throughput performance is not required.

7 Conclusion

In this work, we have compared the message throughput of the FioranoMQ, SunMQ, and WebsphereMQ Java messaging system (JMS) server under various conditions. We first gave a short introduction into JMS and reviewed related work. We presented the testbed and explained our measurement methodology. Then, we presented our experiments and results that we used to develop performance models for the server throughput. We briefly summarize our major findings.

(1) The throughput of the three investigated server types spans over several orders of magnitude with FioranoMQ achieving the highest one and WebsphereMQ achieving the lowest one.

(2) The throughput is significantly larger in the non-persistent mode than in the persistent mode. The difference depends on the server type.

(3) The server throughput depends on the replication grade of the messages and the number of installed filters. FioranoMQ can handle simple correlation ID filters more efficiently than application property filters while SunMQ and WebsphereMQ require the same filtering effort for both filter types.

(4) The message throughput is limited either by the processing logic for small messages or by the transmission capacity for large messages.

(5) The number of configured topics hardly affects the overall capacity of the server.

(6) Complex OR-filters allow a larger message throughput than an equivalent number of simple filters. The performance gain depends significantly on the server type.

(7) The complexity of AND-filters reduces the message throughput for FioranoMQ and SunMQ and the position of the filter components matters, which can be used to optimize the formulation of filter rules. In contrast, WebsphereMQ requires the same time to process a message regardless of the filter complexity and the position of the filter components.

Subsequently, we studied the joint impact of filters and the message replication grade. We designed rather complex experiment series whose measurement results showed the influence of the relevant parameters so well that we could find a quite accurate mathematical approximation model of the message processing time for each server type. These models predict the message throughput for specific application scenarios depending on the average message replication grade, the overall number of installed filters, and the number of different filters. They made it evident that all server types have a basically different performance behavior. Finally, we illustrated the use of the performance models by assessing the suitability of the server types in four simple application scenarios. FioranoMQ led to the highest throughput if filtering is not required; otherwise SunMQ performed better. In contrast, WebsphereMQ cannot be viewed as a high performance JMS solution, but it is rather an allround server platform with many different features including JMS.

References

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

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

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

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

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

[6] Y. Liu and B. Plale, “Survey of Publish Subscribe Event Systems,” Technical Report, No. TR574, Indiana University, May 2003.

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, “Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service,” in 19^{th} *ACM Symposium on Principles of Distributed Computing (PODC)*, July 2000.

[8] G. M”uhl, “Generic Constraints for Content-Based Publish/Subscribe,” in 9^{th} *International Conference on Cooperative Information Systems (CoopIS)*, (London, UK), pp. 211–225, 2001.

[9] Z. Ge, P. Ji, J. Kurose, and D. Towsley, “Min-Cost Matchmaker Problem in Distributed Publish/Subscribe Infrastructures,” in *OpenSig Workshop, From Signalling to Programming*, 2002.

[10] H. Liu and H.-A. Jacobsen, “Modeling Uncertainties in Publish/Subscribe Systems,” in 20^{th} *International Conference on Data Engineering (ICDE)*, (Washington, DC, USA), 2004.

[11] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, “Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems,” *ACM SIGMOD Record*, vol. 30, pp. 115–126, June 2001.

[12] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, “Efficient filtering in publish-subscribe systems using binary decision diagrams,” in 23^{rd} *International Conference on Software Engineering (ICSE)*, (Washington, DC, USA), pp. 443–452, IEEE Computer Society, 2001.

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

[15] A. Carzaniga and A. L. Wolf, “A Benchmark Suite for Distributed Publish/Subscribe Systems,” tech. rep., Software Engineering Research Laboratory, Department of Computer Science, University of Colorado, Boulder, Colorado, 2002.

[16] M. Pang and P. Maheshwari, "Benchmarking Message-Oriented Middleware - TIB/RV vs. SonicMQ," in *Workshop on Foundations of Middleware Technologies, International Symposium on Distributed Objects and Applications (DOA) 2002*, (University of California, Irvine, CA), Nov. 2002.

[17] S. Chen and P. Greenfield, "QoS Evaluation of JMS: An Empirical Approach," in 37^{th} *Annual Hawaii International Conference on System Sciences (HICSS)*, (Washington, DC, USA), IEEE Computer Society, 2004.

[18] U. Farooq, E. W. Parsons, and S. Majumdar, "Performance of Publish/Subscribe Middleware in Mobile Wireless Networks," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 1, pp. 278–289, 2004.

[19] Krissoft Solutions, "JMS Performance Comparison," tech. rep., 2004. `http://www.fiorano.com/comp-analysis/jms_perf_comp.htm`.

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

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

[22] Crimson Consulting Group, "High-Performance JMS Messaging," tech. rep., 2003. `http://www.sun.com/software/products/message_queue/wp_JMSperformance.pdf`.

[23] S. Godard, *Sysstat Monitoring Utilities*, Feb. 2004.