Throughput Performance of the ActiveMQ JMS Server

Robert Henjes, Daniel Schlosser, Michael Menth, and Valentin Himmler

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

Abstract. Communication among distributed software components according to the publish/subscribe principle is facilitated by the Java messaging service (JMS). JMS can be used as a message routing platform if the subscribers install filter rules on the JMS server. However, it is not clear whether its message throughput is sufficient to support large-scale systems. In this paper, we investigate the capacity of the high performance JMS server implementation ActiveMQ. 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 a model for the message processing time at the server and validate it by measurements. This model takes the number of installed filters and the replication grade of the messages into account and predicts the overall message 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]. A salient feature of JMS is that applications can communicate with each other without knowing their communication partners as long as they agree on a uniform 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.

In the non-durable and persistent mode, JMS servers efficiently deliver messages reliably to subscribers that are presently online. Therefore, they are suitable as backbone solution for large-scale realtime communication between loosely coupled software components. For example, some user devices may provide presence information to the JMS. Other users can subscribe to certain message types, e.g. the presence information of their friends' devices. For such a scenario, a high performance routing platform needs filter capabilities and a high capacity to be scalable to a large number of 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 investigate the performance of the ActiveMQ [2] JMS server implementation. We evaluate the maximum throughput by measurement under various conditions. In particular, we consider different numbers of publishers, subscribers, and

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

filters, different kinds of filters, and filters of different complexity to characterize the throughput performance of the ActiveMQ JMS Server. We also propose a mathematical model depending on the number of filters and the message replication grade to approximate the processing time of a message for the ActiveMQ server.

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 and proposes a model for the processing time of a simple message depending on the server configuration. These models are useful to predict the server throughput for specific application scenarios. Finally, we summarize our work in Section 5.

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) is one possible standard of this message exchange. So-called publishers connect to the JMS server and send messages to it. So-called subscribers connect to the JMS server and consume available messages or a subset thereof. So 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 [1] and the JMS server decouples them by acting as a broker. As a consequence, publishers and subscribers do not need to know each other.

The JMS offers two different connection modes: a durable and a non-durable connection type. If a subscriber connects in the durable mode, the messages will be stored for delivery if this client disconnects. All stored messages will be delivered when the client connects next time to the JMS server. Persistence is another option for JMS. If the persistent option is set, each message has to be delivered reliably to all actively connected clients, which is ensured by confirming reception with acknowledgments. In the non-persistent mode the JMS server must deliver the message only with an at-most-once guarantee. This means that the message can be lost, but it must not be delivered twice according to [1]. In this study, we only consider the persistent but non-durable mode.

Information providers with similar themes may be grouped together by making them 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 due to the fact that publishers and subscribers have to know which topics they need to connect to. This results in a slight loose of the decoupling feature in the publish/subscribe context. In addition, topics need to be configured on the JMS server before they can be used actively. If no topics are explicitly introduced at the JMS server, exactly one default topic is present, to which all subscribers and publishers are connected.

Filters are another option for message selection. A subscriber may install a message filter on the JMS server. Only the messages matching the filter rules are forwarded to the

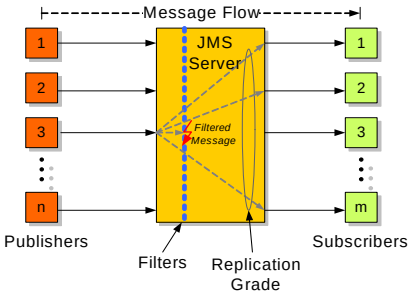


Fig. 1. The JMS server delivers messages from the publishers to all subscribers with matching filters.

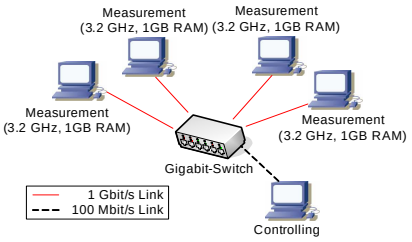


Fig. 2. Testbed environment.

respective subscriber instead of all messages. In contrast to topics, filters are installed dynamically during the operation of the server by each subscriber.

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 fixed header of JMS messages as the only user definable option within this header section. Correlation ID filters try to match these IDs whereby wildcard filtering is possible, e.g., in the form of ranges like [#7;#13], which means all IDs between #7 and #13 are matched including #7 and #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 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 they require 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 [5], SonicMQ [6], TibcoEMS [7], and WebsphereMQ [8]. The study focuses on several message modes, e.g., durable, persistent, etc., but it does not consider filtering, which is the main objective in our work. The authors of [9] conduct a benchmark comparison for the SunMQ [10] and IBM WebsphereMQ. They tested throughput performance in various message modes and, in particular, with different acknowledgement options for the persistent message mode. They also examined simple filters, but they did not conduct parametric studies, and no performance model was developed. The objective of our work is the development of such a performance model to forecast the maximum message throughput for given application scenarios. A proposal for designing a “Benchmark Suite for Distributed Publish/Subscribe Systems” is presented in [11] but without measurement results. The setup of our experiments is in line with these recommendations. General benchmark guidelines were suggested in [12] 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 [13] but without validation by measurements. The authors of [13] present in [14] an enhanced framework to analyze and simulate a publish/subscribe system. In this work also filters are modeled as a general function of time but not analyzed in detail. The validation of the analytical results is done by comparing them to a simulation. In contrast, our work presents a mathematical model for the throughput performance in the non-durable mode including several filter types and our model is validated by measurements on an existing implementation of a JMS server. Several other 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, intelligent optimizations may be applied to reduce the filter overhead [15].

The Apache working group provides the generic test tool JMeter for throughput tests of the ActiveMQ [16]. However, it has only limited functionality such that we rely on an own implementation to automate our experiments.

3 Test Environment

Our objective is the assessment of the message throughput of the ActiveMQ JMS server by message 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 2. Four of them are production machines and one is used for control purposes, e.g., controlling jobs like setting up test scenarios and monitoring 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 2048 MB system memory. Their operating system is SuSe Linux 9.1 with kernel version 2.6.5-smp installed in standard configuration. The “smp”-option enables the support of the hyper-threading feature of the CPUs. Hyper-threading means that a single-core-CPU uses multiple program and register counters to virtually emulate a multi-processor system. In our case we have two virtual cores. To run the JMS environment we installed Java JRE 1.5.0 [17], 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. Our test application is designed such that JMS subscribers or publishers can run as Java threads. Each thread has an exclusive connection to the JMS server component and represent a so-called JMS session. A management thread collects the measured values from each thread and appends these data to a log file in periodic intervals.

In our test environment 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 Server Installation

The ActiveMQ server version 4.0 stable [2] is an open source software provided by the Apache group. We installed it on one of the above described Linux machines in default configuration such that the hyper-threading feature of the Linux kernel is used and the internal flow control is activated. To ensure that the ActiveMQ JMS server has enough buffer memory to store received messages and filters we set explicitly the memory for the Java Runtime Environment to 1024 MB.

3.3 Measurement Methodology

Our objective is the measurement of the JMS server capacity and we use the overall message throughput of the JMS server machine as performance indicator. We keep the server in all our experiments as close as possible to 100% CPU load. We verify that no other resources on the server machine like system memory or network capacity are bottlenecks. The publisher and subscriber machines must not be bottlenecks. Therefore their CPU loads must be lower than 75%. To monitor these side conditions, we use the information provided in the Linux „/proc" path. We monitor the CPU utilization, I/O, memory, and network utilization for each measurement run. Without a running server software, 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 95%.

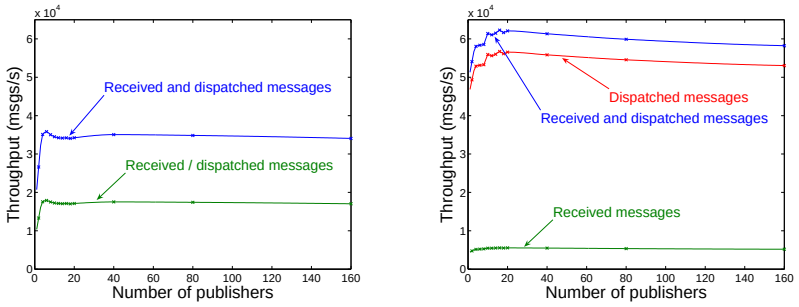
Experiments are conducted as follows. The publishers run in a saturated mode, i.e., they send messages as fast as possible to the JMS server. However, they are slowed down by the flow control of the server such that we observe publisher-side message queueing. We count the overall number of sent messages at the publishers and the overall number of received messages by the subscribers to calculate the server's rate of received and dispatched messages. Our measurement runs take 10 minutes whereby we discard the first seconds due to warmup effects. 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. Therefore, we omit them in the figures of the following sections. The following experiments use the non-durable and persistent messaging mode as described in the Section 2.

4 Measurement Results

In this section we investigate the maximum message throughput of the ActiveMQ JMS server. The objective is to assess and characterize the impact of specific application scenarios on the 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 and Subscribers

In our first experiment, we study the impact of different numbers of publishers and subscribers on the message throughput. The results of the following two experiments yield the minimum number of clients which have to be connected to the JMS server to fully load the JMS server. In the persistent mode, i.e., lost messages are retransmitted by the JMS server and messages are preliminarily written to a disk for recovery purposes. Also each message is explicitly acknowledged by the recipient of the message.



(a) 1 subscriber thread ($r = 1$) (b) 10 subscriber threads ($r = 10$)

Fig. 3. Impact of the number of publishers on the message throughput

Impact of Number of Publishers We study the impact of the number of publishers for a single connected subscriber without filters and for 10 connected subscribers without filters. We present the throughput of the messages that are received and dispatched by the server in Figure 3(a) and Figure 3(b) together with their sum which we call the overall message throughput in the following.

For a single subscriber, the throughput of received messages equals the one of dispatched messages since each message is forwarded to only one subscriber while for 10 subscribers, the dispatched throughput is 10 times larger than the received throughput. As the overall throughput of the server is limited, the received throughput for 10 subscribers is clearly smaller than the one for a single subscriber. Thus, the average number of replications of each message clearly impacts the received throughput and we call it the replication grade r in the following. A comparison of the absolute throughput of both experiments shows that only a relatively small overall throughput of 34000 msgs/s can be achieved for a single subscriber while for 10 subscribers, a maximum overall throughput of 62000 msgs/s can be achieved for 16 publishers.

For both experiments, the throughput is almost independent of the number of publishers if this number is sufficiently large (about 10) such that we use at least 10 subscribers in the following experiments. We also observed that the server cannot be fully loaded with a single subscriber.

Impact of the Number of Subscribers Similarly to the experiment above, we investigate the impact of the number of subscribers on the JMS server throughput. To that end, we have 20 publisher threads running on one machine and vary the number of subscribers on two other machines. Figure 4 shows the received, dispatched, and the overall message throughput. The maximum overall throughput of about 60000 msgs/s is reached for 10 connected subscribers and decreases decreases only slightly for an increased number of subscribers. The received throughput of the JMS server decreases with an increasing number of subscribers. We observe a received throughput of about 16000 msgs/s for one subscriber and of about 130 msgs/s for 320 subscribers.

Unlike in Figure 3(a) or Figure 3(b), the received message rate decreases significantly with an increasing number of subscribers m. This can be explained as follows. No filters are applied and all messages are delivered to all subscribers such that each message is replicated m times. We call this a replication grade $r = m$. This requires

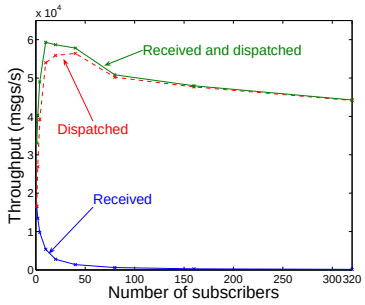


Fig. 4. Impact of the number of subscribers on the message throughput (20 publishers).

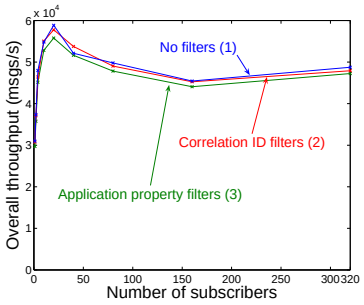


Fig. 5. Impact of filter activation and the number of subscribers on the message throughput.

more CPU cycles for dispatching messages and increases the overall processing time of a single received message. As a consequence the rate of received messages at the server decreases. Thus, the replication grade has to be considered when performance measures from different experiments are compared.

4.2 Impact of Filter Activation

We evaluate the impact of filter activation on the message throughput. We perform 3 different experiment series where all publishers send messages with an application property or correlation ID value set to #0. We install 20 publisher threads on a single publisher machine and a varying number of m subscriber threads on one subscriber machine. We use the following filter configurations which lead to a message replication grade of $r = m$.

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

the number of subscribers on the message

Figure 5 illustrates the overall message throughput for the above described experiments. The overall message throughput differs only slightly for the three different experiments. For other server types, like Bea WebLogic [18] or FioranoMQ [5], filter activation results in a decreased overall throughput compared to experiment (1). A comparison of the results for experiment (2) and (3) shows that correlation ID filters lead to a slightly larger throughput than application property filters for the ActiveMQ server. In the following experiments we focus only on application property filters because they are more flexible.

From Figure 5 we can also conclude, that an increasing number of equal filters has almost no impact on the overall throughput for more than 300 installed filters. Since the overall throughput remains almost constant at a value of 50000 msgs/s.

4.3 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.

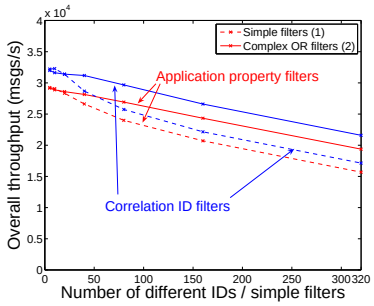


Fig. 6. Impact of simple filters and complex OR-filters on the message throughput for a replication grade of $r=1$.

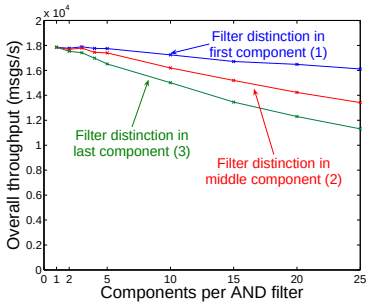


Fig. 7. Impact of an early non-match decision for AND-filters on the message throughput depending on the filter complexity for a replication grade of $r=1$.

These two options are alternative filter configurations for the same application scenario. We assess the JMS server performance for both options by two different experiments, which have both a message replication grade of $r=1$ if 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 $\#(j \cdot \lfloor \frac{n}{5} \rfloor + i)$ with $i \in [1; \frac{n}{5}]$ using an OR-filter.

We use in this experiment one publisher machine with 20 publisher threads and one subscriber machine with a varying number of subscribers for the simple filters approach or 5 subscribers for the complex or filters experiment, respectively. We also repeat the experiment for correlation ID and application property filters.

Figure 6 shows the message throughput depending on on the number of different IDs n in the complex filter. The throughput for the complex OR filters and the simple filters is for both different filter types in the same order of magnitude. The throughput for the simple filter experiment (1) is mostly lower than the throughput for the complex OR filters experiment (2). Also from this experiment we can conclude, that throughput performance of the application property filters and the correlation ID filters only slightly differ.

4.4 Impact of Complex AND-Filters

In the application header section of a message, multiple properties, e.g. $P_1, ..., P_k$, 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. They are only applicable for application property filtering. We use one machine with 20 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:$P_1 = \#j, P_2 = \#0, ..., P_n = \#0$

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

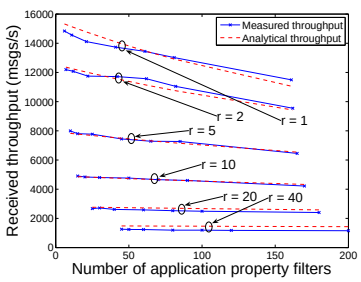
Figure 7 shows the message throughput depending on the filter complexity n. In all scenarios, the filter complexity reduces the server capacity. Experiment (1) yields the largest message throughput, followed by experiment (2) and (3). The evaluation of a complex filter is obviously stopped as soon as a mismatch of one of the components occurs and the evaluation is performed from left to right of its components, as required by the JMS API [19]. precedence level. Parentheses can be used to change this order. This early reject decision of the filters reduces 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.

4.5 Joint Impact of the Number of Filters and the Replication Grade

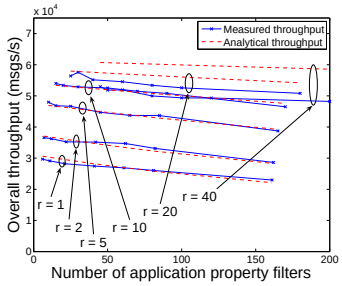
We study the impact of the replication grade r, the number of filters n_{fltr} on the message throughput of the JMS server.

Experimental Analysis The publishers send only messages with ID #0 as a property in the application property part. To achieve a replication grade of r, we set up r different subscribers with a filter for ID #0. Furthermore, we install m_{subs}^{add} additional subscribers, each installing a filter for ID #1. So we have a total of $m_{subs}^{add} + r$ subscribers installed and the number of filters in the system is $n_{fltr} = m_{subs}^{add} + r$. We use the following values for our experiments $r \in \{1, 2, 5, 10, 20, 40\}$, $m_{subs}^{add} \in \{5, 10, 20, 40, 60, 80, 160\}$, and conduct them with 20 publisher threads on one publisher machine and with a variable number of $r + m_{subs}^{add}$ subscribers equally distributed over two subscriber machines.

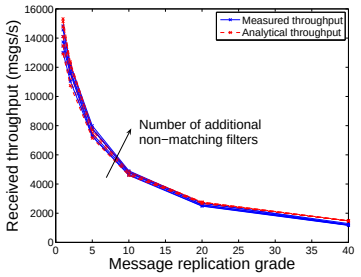
The solid lines in Figure 8(a) and Figure 8(b) show, that the measured received and overall throughput slightly decreases for an increasing number of installed filters n_{fltr} for the above described experiments. The message throughput is also clearly influenced by the message replication grade r. Therefore, we provide in Figure 8(c) and Figure 8(d) an alternative presentation of the same data with the replication grade on the x-axis and separate curves for the number of additional non-matching filters. Figure 8(c) and Figure 8(d) show that the received throughput clearly decreases and the overall throughput clearly increases with an increasing replication grade. Comparing Figure 8(c) and Figure 8(d) with Figure 8(a) and Figure 8(b) leads to the conclusion that the impact of the



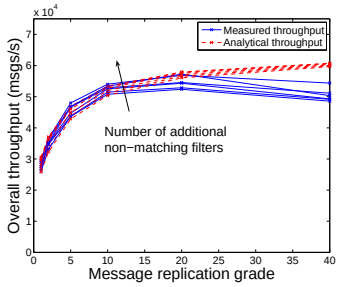
(a) Impact of the number of filters on the received throughput



(b) Impact of the number of filters on the overall throughput



(c) Impact of the replication grade on the received throughput



(d) Impact of the replication grade on the overall throughput

Fig. 8. Impact of the number of application property filters and the replication grade on message throughput of the ActiveMQ server

message replication grade on the message throughput is larger than the impact of the number of installed filters. The throughput is the same regardless if we use equal or different filters that do not match. Which is unlike with the SunMQ [20] which leads to a different analytical model. We conducted the same experiment for correlation ID filters and obtained very similar results.

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 time overhead t_{rcv} which is almost independent of the number of installed filters.
- a fixed time overhead $n_{fltr} \cdot t_{fltr}^{all}$ caused by the JMS server due to the number of all installed filters. This value depends on the application scenario.
- a 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:

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

From previous study it is known, that this model holds also for other JMS server implementations, e.g. the FioranoMQ [21] and the Bea WebLogic [18]. Within time B,

one message is received and r messages are sent on average. Therefore, the received and overall throughput are given by $\frac{1}{B}$ and $\frac{r+1}{B}$, respectively. The values for t_{rcv}, t_{fltr}, and t_{tx} can be derived from the measured received message throughput in our experiments by least square approximation.

Validation of the Model by Measurement Data The curves for replication grade $r=20$ and $r=40$ do not follow the trend of the curves for replication grades $r\in\{1,2,5,10\}$. Therefore, we respect only the experiments with replication grades $r\in\{1,2,5,10\}$ in the least squares approximation and obtain the model parameters $t_{rcv}=4.9\cdot10^{-5}$s, $t_{fltr}=1.6\cdot10^{-7}$s, and $t_{tx}=1.5\cdot10^{-5}$s. We use these parameters to calculate the analytical throughput which is illustrated in Figures 8(a)–8(d) by dashed lines. For small replication grades $r=\{1,...,10\}$ the analytical throughput is in good accordance with the measured throughput. We also measured the performance of the correlation ID filters. But we measured only slightly different message throughput values and we can therefore omit a rather complex model as for example used for the SunMQ [20] server. As mentioned above, the capacity curves for message replication grades $r=20$ and $r=40$ in Figure 8(b) are lower than expected from an intuitive extrapolation of the other curves. The reason for this observation might be a maximum internal transmission capacity of the server such that the server CPU is no longer the limiting criterion. In previous studies with other server types we have not encountered such a phenomenon since the overhead of these servers for message filtering was significantly larger than the one for ActiveMQ. As a consequence, the transmission capacity of these servers was sufficient even for a large message replication grade of $r=40$. However, we expect to observe similar saturation effects for these servers, too, if we further increase the message replication grade in this experiment series.

5 Conclusion

In this work, 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. Then we have investigated the maximum message throughput of the Java Messaging System (JMS) server "ActiveMQ" under various conditions.

We studied the server capacity depending on the number of publishers and subscribers and found out that the maximum server performance can be achieved only if sufficiently many publishers and subscribers participate in the system. Correlation ID filters lead to a slightly larger throughput than application property filters and complex OR-filters are more efficient than equivalent single filters. As ActiveMQ obviously supports an early reject for non-matching AND-filters, practitioners should carefully choose the order of the filter components. We showed that the use of filters has only a small impact on the server capacity. This makes the ActiveMQ server very attractive compared to other JMS server solutions [5], [18] if filtering is heavily used. In contrast, the message replication grade has a significant impact on both the received and overall throughput. We finally developed an analytical model to describe the capacity of the ActiveMQ and provided appropriate model parameters based on our measurements. This model is useful to predict the server capacity in practical application scenarios.

Currently, we investigate the capacity of server clusters, which are intended to increase the overall message throughput of the system. Furthermore, we investigate different options to reduce the overhead for filter processing times.

References

1. Sun Microsystems, Inc.: Java Message Service API Rev. 1.1. (2002) http://java.sun.com/products/jms/.
2. Apache: ActiveMQ, Reference Documentation. (2006) http://www.activemq.org.
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. Fiorano Software, Inc.: FioranoMQ^{TM}: Meeting the Needs of Technology and Business. (2004) http://www.fiorano.com/whitepapers/whitepapers_fmq.pdf.
6. Sonic Software, Inc.: Enterprise-Grade Messaging. (2004) http://www.sonicsoftware.com/products/docs/sonicmq.pdf.
7. Tibco Software, Inc.: TIBCO Enterprise Message Service. (2004) http://www.tibco.com.
8. IBM Corporation: IBM WebSphere MQ 6.0. (2005) http://www-306.ibm.com/software/integration/wmq/v60/.
9. Crimson Consulting Group: High-Performance JMS Messaging. Technical report (2003) http://www.sun.com/software/products/message_queue/wp_JMSperformance.pdf.
10. Sun Microsystems, Inc.: Sun ONE Message Queue, Reference Documentation. (2006) http://developers.sun.com/prodtech/msgqueue/.
11. 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)
12. Wolf, T.: Benchmark für EJB-Transaction und Message-Services. Master's thesis, Universität Oldenburg (2002)
13. Baldoni, R., Contenti, M., Piergiovanni, S.T., Virgillito, A.: Modelling Publish/Subscribe Communication Systems: Towards a Formal Approach. In: 8th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2003). (2003) 304–311
14. Baldoni, R., Beraldi, R., Piergiovanni, S.T., Virgillito, A.: On the modelling of publish/subscribe communication systems. Concurrency - Practice and Experience **17** (2005) 1471–1495
15. Mühl, G., Fiege, L., Buchmann, A.: Filter Similarities in Content-Based Publish/Subscribe Systems. Conference on Architecture of Computing Systems (ARCS) (2002)
16. Apache Incubator: ActiveMQ, JMeter Performance Test Tool. (2006) http://www.activemq.org/jmeter-performance-tests.html.
17. Sun Microsystems, Inc.: JRE 1.5.0. (2006) http://java.sun.com/.
18. Bea Systems: Bea WebLogic Server 9.0. (2006) http://dev2dev.bea.com.
19. Sun Microsystems, Inc.: Java Message Service Specification, Version 1.1. (2002) http://java.sun.com/products/jms/docs.html.
20. Henjes, R., Menth, M., Zepfel, C.: Throughput Performance of Java Messaging Services Using Sun Java System Message Queue. In: High Performance Computing & Simulation Conference (HPC&S), Bonn, Germany (2006)
21. Henjes, R., Menth, M., Gehrsitz, S.: Throughput Performance of Java Messaging Services Using FioranoMQ. In: 13thGI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), Erlangen, Germany (2006)