Throughput Performance of Java Messaging Services Using WebsphereMQ

Robert Henjes, Michael Menth, and Christian Zepfel
Department of Distributed Systems
Institute of Computer Science
University of Würzburg, Germany
Email: {henjes,menth,zepfel}@informatik.uni-wuerzburg.de

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

I. Introduction

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

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

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

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

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

II. Background

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

A. The Java Messaging Service

Messaging facilitates the communication between remote software components. The Java Messaging Service (JMS) standardizes this message exchange. The so-called publishers generate and send messages to the JMS server, the so-called subscribers consume these messages – or a subset thereof – from the JMS server, and the JMS server acts as a relay node [3], which controls the message flow by various message filtering options. This is depicted in Figure 1. Publishers and subscribers rely on the JMS API and the JMS server decouples them by acting as an isolating element. As a consequence, publishers and subscribers do not need to know each other. The JMS offers several modes. In the persistent mode, messages are delivered reliably and in order. To that end, lost messages are retransmitted by the JMS server and messages are preliminarily written on a disk for recovery purposes. 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 only consider the persistent but non-durable mode.

Information providers with similar themes may be grouped together and publish to a so-called common topic; only those

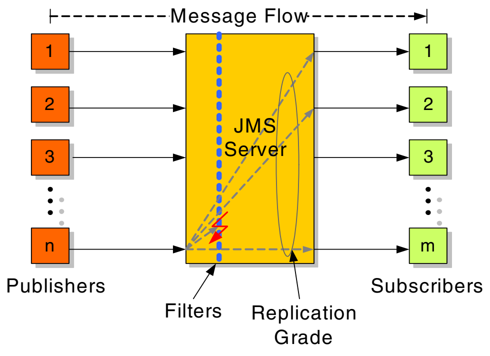

Fig. 1. The JMS server decouples publishers and subscribers.

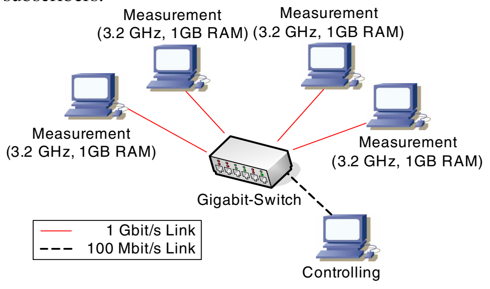
Measurement (3.2 GHz, 1GB RAM)
Measurement (3.2 GHz, 1GB RAM)
Measurement (3.2 GHz, 1GB RAM)
Measurement (3.2 GHz, 1GB RAM)
Gigabit-Switch
1 Gbit/s Link
100 Mbit/s Link
Controlling

Fig. 2. Testbed environment.

subscribers having subscribed for that specific topic receive their messages. Thus, topics virtually separate the JMS server into several logical sub-servers. Topics provide only a very coarse and static method for message selection. In addition, topics need to be configured on the JMS server before system start. Filters are another option for message selection. A subscriber may install a message filter on the JMS server, which effects that only the messages matching the filter rules are forwarded instead of all messages in the corresponding topic. In contrast to topics, filters are installed dynamically during the operation of the server. 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.

B. Related Work

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

III. Test Environment

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

A. Testbed

Our test environment consists of five computers that are illustrated in Figure 2. 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. We installed the the IBM Websphere MQ 6.0 Trial version [2] on the server machine with the default configuration except for the following modifications. For the sake of performance 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.

B. *Measurement Methodology*

Our objective is the measurement of the JMS server capacity. Therefore, we intend to 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 [14]. We monitor the CPU utilization, I/O, memory, and network utilization for each measurement run. Without a running server, the CPU utilization of the JMS server machine does not exceed 2%, and a fully loaded server has a CPU utilization between 90% and 98%. For some reasons these values could not be increased by any means.

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.

IV. Measurement Results

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

A. *Impact of the Number of Publishers*

In our first experiment, we study the impact of the number of publishers on the message throughput. Two machines carry a varying number of publishers and one machine hosts 1 subscriber. Figure 3 shows the message throughput at the JMS server in the persistent mode. The throughput of received and dispatched messages is plotted separately, as well as their sum which we call the overall throughput. The overall message throughput reaches its maximum rate at 2000 msgs/s for 5 or more publishers. Hence, the number of publishers hardly influences the JMS server throughput. As a consequence, we use in the following experiments at least 5 or more publishers. In this particular experiment series, the JMS server could only utilized to 70% due to the limitation of a single subscriber. To assess the impact of the persistent mode, we conduct the same experiment in the non-persistent mode and the results are collected in Figure 4. The overall throughput is about 10000 msgs/s for the non-persistent mode in contrast to about 2000 msgs/s for the persistent mode. Thus, the server capacity has increased by 400%. In this case, the server is utilized to 98%. Due to the non-persistent mode, the dispatched message rate is lower than the received message rate for a large number of publishers which leads to about 8% message loss in the end. We conducted the same experiments with 5 subscribers instead of a single one and achieved an overall message throughput of 6000 msgs/s and 11000 msgs/s, respectively.

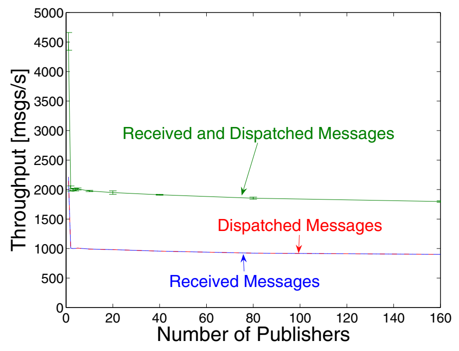

Fig. 3. Impact of the number of publishers on the message throughput in the persistent mode.

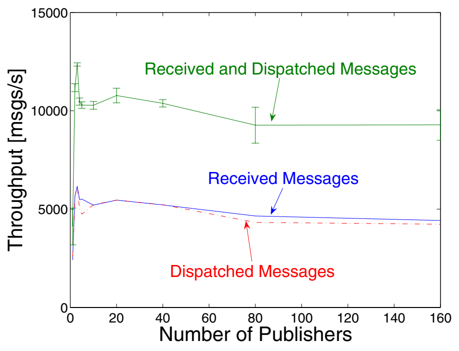

Fig. 4. Impact of the number of publishers on the message throughput in the non-persistent mode.

We also observe on our monitoring tool that the CPU utilization of the server machine is very low if we install just 2 publishers. For 4 or more publishers, the CPU utilization reaches a saturation and cannot be further increased. Thus, at least 4 publishers are needed to fully load the JMS server. We repeated the experiment series several times and calculated the 99.99% confidence intervals on this basis. They are shown in the figure for the overall throughput. Obviously, they are very narrow which results from hardly varying results and we omit them in the following figures.

B. Impact of the Number of Subscribers

Similarly to the above, we investigate the impact of the number of subscribers on the JMS server throughput. To that end, we have 5 publishers threads running on one machine and vary the number of subscribers on two other machines. Figure 5 shows the received, dispatched, and the overall message throughput. The overall throughput of the JMS server starts at 2000 msgs/s for 1 subscriber, it increases with an increasing number of subscribers to a value of about 11000 msgs/s for 40 subscribers, and it decreases then to 6000 msgs/s for many subscribers. Thus, the previous experiments led to a untypically low capacity due to the single subscriber.

Unlike in Figure 3, the received message rate decreases significantly with an increasing number of subscribers n. This can be explained as follows. No filters are applied and all messages are delivered to any subscriber. Thus, each message is replicated n times and we call this a replication grade of $r = n$. This requires more CPU cycles for dispatching messages and increases the overall processing time of a single message. As a consequence, the received message rate is reduced because the overall throughput capacity of the server stays constant. Hence, the replication grade must be considered when performance measures from different experiments are compared.

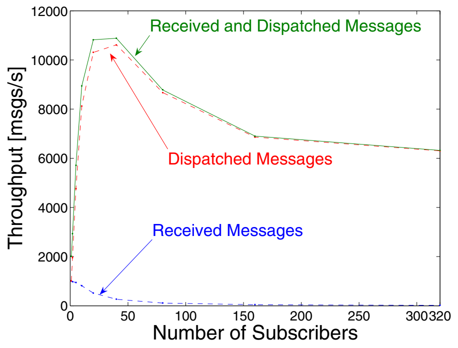

Fig. 5. Impact of the number of subscribers on the message throughput.

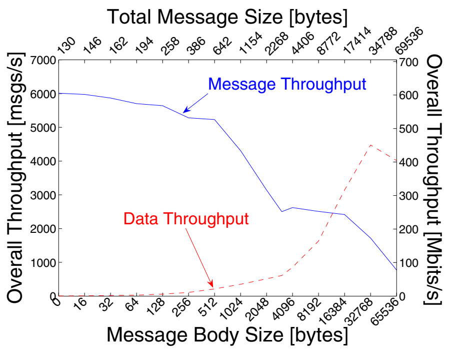

Fig. 6. Impact of the message body size on the message and data throughput.

C. Impact of the Message Size

The throughput of a JMS server can be measured in messages per second (message throughput) or in transmitted data per second (data throughput). The message body size has certainly an impact on both values. We test the maximum throughput depending on the message size. We set up 5 publishers on one publisher machine and 5 subscribers on a single subscriber machine without any filters; a single subscriber is not able to fully utilize the server CPU. Figure 6 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 6 shows that an increasing message body size decreases the message throughput and increases the data throughput significantly. For small message bodies, the message throughput is limited by 6000 msgs/s while for very large message sizes, the data throughput increases significantly up to 450 Mbit/s. Obviously, the network interface of the JMS server becomes the system bottleneck. We proved this speculation by measuring the maximum throughput of a single TCP connection which amounts to at most 350 Mbit/s in one direction. In our experiments, the default value for the message body size is 0 bytes.

D. Impact of Filter Activation

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

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

We use 5 publisher threads on a single publisher machine and a varying number of n subscriber threads on two subscriber machines. Figure 7 illustrates the overall throughput for the above described experiments. In all experiments, correlation ID and application property filters lead to the same throughput. The curves for experiment (1) and (2) show the same high throughput, which is due to the same replication grade of $r = n$. However, if we change the replication grade

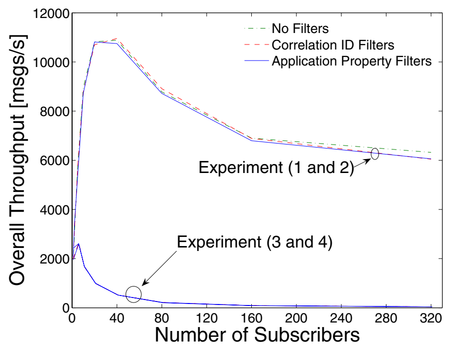

Fig. 7. Impact of filter activation and the number of subscribers on the message throughput for correlation ID filters.

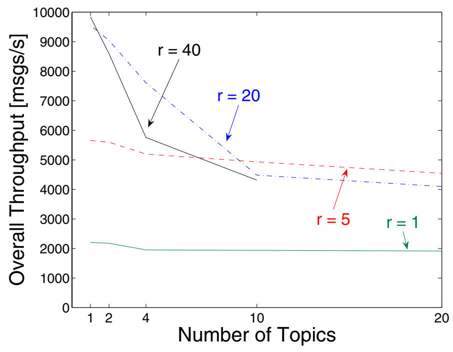

Fig. 8. Impact of the number of topics on the message throughput for different replication grades.

to $r = 1$ in (3) by turning all filters but one to #1 instead to #0, we see a clear reduction of the throughput. Hence, the message replication grade r has a significant impact on the overall throughput. A comparison of the results for experiment (3) and (4) concludes that different filters lead to the same throughput as equal filters. Obviously, WebsphereMQ does not implement optimized filter matching as the JMS server cannot take advantage of equal filters to save processing power per message. More on filter optimization can be found in [13].

E. Impact of Topics

Messages published to a specific topic are only dispatched to consumers who have subscribed to this particular topic. Thus, topics allow a very coarse form of message selection. In this section, we evaluate the impact of the number of topics on the message throughput for different replication grades. In our next experiment, 5 publisher threads are installed on one publisher machine and two machines host the subscribers. We vary the number of topics on the JMS server. Each publisher is connected to every topic and sends messages to them in a round robin manner. A replication grade r is obtained by registering r subscribers for each topic. Figure 8 shows that the message throughput remains constant for a replication grade of $r = 1$, independently of the number of topics. The overall throughput is so low due to the single subscriber per topic, which is consistent with the results in Section IV-B. For larger replication grades, the throughput is significantly larger, it decreases for an increasing number of topics, and converges to a value of around 5000 msgs/s.

F. 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 or
(2) with a single but complex OR-filter searching for all desired message types.

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

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

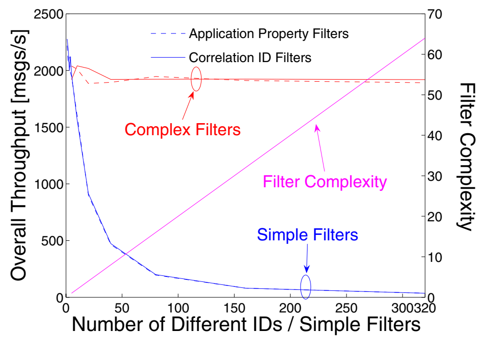

Fig. 9. Impact of simple filters and complex OR-filters on the message throughput for a replication grade of $r = 1$.

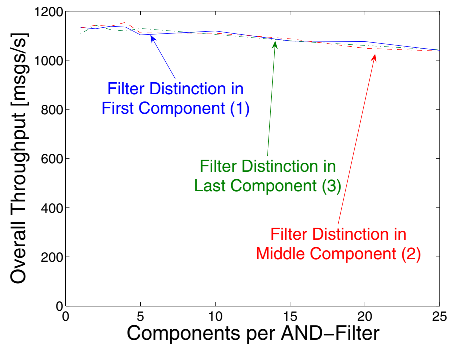

Fig. 10. Impact of an early non-match decision for AND-filters on the message throughput depending on the filter complexity for a replication grade of $r = 1$.

We use in this experiment one publisher machine with 5 publisher threads and one subscriber machine with a varying number of simple subscribers or 5 complex subscribers, respectively. Figure 9 shows the message throughput and the filter complexity depending on the number of different IDs

n. The diagonal line indicates the length of the complex OR-filters. Complex filters (1) lead to significantly more throughput than the equivalent number of simple filters per client (2). Thus, it is better to use complex OR-filters than to filter each component separately by an additional subscriber. This observation holds both for application property and for correlation ID filters.

G. 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. Note that complex AND-filters are only applicable for application property filtering. In the following, we use one machine with 10 publisher threads and one machine with $m=10$ subscriber threads that are numbered by $j \in [1;m]$. We design three experiment series with a different potential for optimization of filter matching. The subscribers set up the following complex AND-filters of different length n:

(1) for subscriber j: $P_1 = \#j, P_2 = \#0, ..., P_n = \#0$
(2) if n is odd:
for subs. j: $P_1 = \#0, ..., P_{\frac{n+1}{2}-1} = \#0, P_{\frac{n+1}{2}} = \#j, P_{\frac{n+1}{2}+1} = \#0, ..., P_n = \#0$
if n is even:
for subs. j if $j \leq \frac{n}{2}$: $P_1 = \#0, ..., P_{\frac{n}{2}-1} = \#0, P_{\frac{n}{2}} = \#j, P_{\frac{n}{2}+1} = \#0, ..., P_n = \#0$
for subs. j if $j > \frac{n}{2}$: $P_1 = \#0, ..., P_{\frac{n}{2}} = \#0, P_{\frac{n}{2}+1} = \#j, P_{\frac{n}{2}+2} = \#0, ..., P_n = \#0$
(3) 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 reject non-matching messages by looking at the first component in experiment (1), at the first half of the components in experiment (2), and at all n components in experiment (3). This experiment is designed such that both the replication grade and the number of subscribers is constant and that only the filter complexity n varies. To avoid any impact of different message sizes in this experiment series, we define $k=25$ properties in all messages to get the same length. Figure 10 shows the message throughput depending on the filter complexity n. In all scenarios, the filter complexity slightly reduces the server capacity, but there is no significant difference regarding the server throughput. Thus, no advantage is taken of the potential for an early rejection of filters to shorten the processing time of a message and to increase thereby the server capacity.

H. An Analytical Model for the Message Throughput

We have learned from Section IV-D that both the number of filters and the replication grade impact the JMS server capacity. In this section, we investigate their joint impact and present a simple model to forecast the server performance for a given number of filters and for an expected replication grade. The model is validated by measurements.

1) Joint Impact of the Number of Filters and the Replication Grade: We set up experiments to conduct parameter studies regarding the number of installed filters and the replication grade r of the messages. We use one publisher and one subscriber machine. Five publishers are connected to the JMS server and send messages with application property value #0 in a saturated way. Furthermore, $n+r$ subscribers are connected to the JMS server, r of them filter for #0 while the other n subscribers filter for different values. 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.

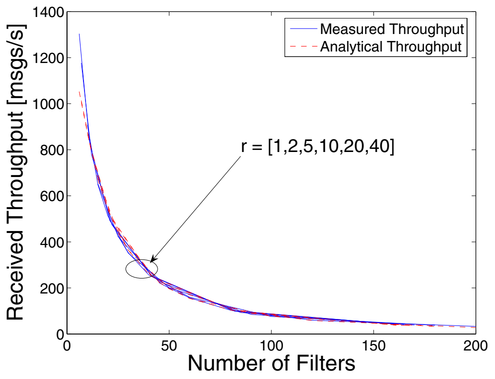

Fig. 11. Impact of the number of filters n_{fltr} and the message replication grade r on the received message

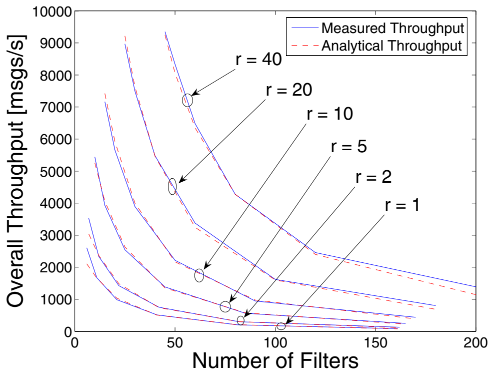

Fig. 12. Impact of the number of filters n_{fltr} and the message replication grade r on the overall message throughput – measurements and analytical data.

Figure 11 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 message throughput of the system which is obviously independent of the replication grade. Figure 12 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.

2) A Simple Model for the Message Processing Time: Figure 11 shows that the message processing time depends only on the number of filters n_{fltr} but not on the replication grade r. Thus, the time to send messages is obviously so small for the server that it is not noticeable for a replication grade of up to $r=40$. A linear approximation model for the message processing time regarding the number of filters does not work. Therefore, we propose the following model for the message processing time B:

$$B = t_{rcv} + n_{fltr} \cdot \sqrt{(n_{fltr})} \cdot t_{fltr} \quad (1)$$

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}.

3) Validation of the Model by Measurement Data: Within time B, one message is received and r messages are dispatched by the server. Thus, the received and overall message throughput is given by $\frac{1}{B}$ and $\frac{r+1}{B}$, respectively, and corresponds to the results in Figures 11 and 12. The parameter n_{fltr} for the message processing time B is known from the respective experiments. We fit the parameters t_{rcv} and t_{fltr} by a least squares approximation [15] to adapt the model in Equation (1) to the measurement results. We get the best fit for $t_{rcv} = 7.03 \cdot 10^{-4}$ 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 (1), and plot them with dashed lines in Figures 11 and 12. 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. This is useful for the dimensioning of distributed systems. In addition, the parameters t_{rcv} and t_{fltr} allow a simple comparison of the server capacity on different server platforms.

V. Conclusion

In this work, we have investigated the capacity of the Java Messaging System (JMS) server WebsphereMQ under various conditions. We first gave a short introduction into JMS and reviewed related work. We presented the testbed and explained our measurement methodology. We performed many parametric experiment series and obtained measurement results. We briefly summarize our findings.

(1) At least 5 subscribers, and 5 publishers sending in a saturated mode are required to fully utilize the server CPU and to obtain the server's maximum overall message throughput.
(2) In the non-persistent mode, the JMS server achieves a significantly larger throughput but it may lose messages.
(3) The message size has a significant impact on the message and the data throughput of the server.
(4) The number of topics hardly influences the server capacity.
(5) The message replication grade and the number of filters have a large impact on the server capacity.
(6) Application property and correlation ID filters lead to the same message throughput.
(7) Different and equal filters impose the same effort for filtering on the JMS server.
(8) Complex OR-filters lead to a significantly higher throughput than the equivalent number of simple filters per client.
(9) For complex AND-filters, the order of the filter components has no impact on the server capacity.

Finally, we studied the joint impact of the number of filters n_{fltr} and the replication grade r of the messages on the overall message throughput and developed a model to predict the overall message throughput depending on n_{fltr} and r. It is useful to predict the server capacity in practical application scenarios and for dimensioning purposes.

Currently, we investigate the message throughput of other JMS servers than the WebsphereMQ to compare their capacity [16]. We study the message waiting time taking into account the variability of the replication grade in practice. In addition, we intend to increase the JMS throughput by the use of server clusters.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[16] R. Henjes, M. Menth, S. Gehrsitz, and C. Zepfel, "Throughput Performance of Popular JMS Servers," in *ACM SIGMETRICS (short paper)*, Saint-Malo, France, June 2006.