NOTICE: This is the author's version of a work accepted for publication by Springer. Changes resulting from the publishing process, including editing, corrections, structural formatting and other quality control mechanisms, may not be reflected in this document. Changes may have been made to this work since it was submitted for publication in ITG/GI Symposium Communication in International Teletraffic Congress (ITC), 2007. The final publication is available at Springer via http://dx.doi.org/ 10.1007\/978-3-540-72990-7_20.

Impact of Complex Filters on the Message Throughput of the ActiveMQ JMS Server

Robert Henjes, 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,menth,himmler}@informatik.uni-wuerzburg.de

Abstract. In this paper we investigate the maximum message throughput of the ActiveMQ server in different application scenarios. We use this throughput as a performance criterion. It depends heavily on the installed filters and the message replication grade. In previous work, we have presented measurement results and an analytical model for simple filters. This work extends these studies towards more complex configuration options. It provides measurement results and analytical performance models for complex AND-, OR-, and IN-filters. The results are useful to understand the performance of JMS servers and help to dimension large distributed JMS-based systems.

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 types of messages 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.

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

In previous work we have measured and modelled the message throughput of the ActiveMQ server depending on the number of installed simple filters n_{fltr} and the replication grade r of the messages. OR- and AND-filters are more complex as they may have different numbers of filter components. We also observed that the message throughput of the server decreases significantly with an increasing length of these complex filters. In this paper, we design suitable experiment series, perform a large number of measurements, and extend the previously found model to cover the server behavior in the presence of complex filters. The formula is still simple and can be used by engineers to predict the server performance for special use cases.

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 develops the experiment design, shows measurement results, and Section 5 proposes a model for the processing time of a simple message depending on the server configuration and validates it by the obtained measurement data. Finally, we summarize our work in Section 6.

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 [2], which controls the message flow by various message filtering options. This architecture 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. In the non-durable mode, messages are forwarded only to subscribers who are presently online. 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

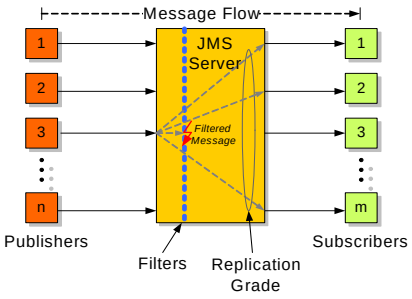


Fig. 1. The JMS server delivers messages from the publishers to all subscribers with matching filters.

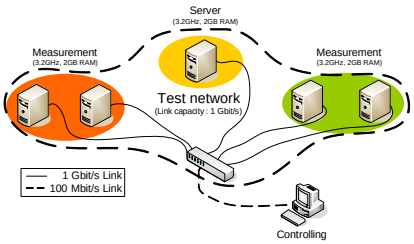


Fig. 2. Testbed environment.

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 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 fixed header, a user defined property header section, and the message payload itself [1]. So-called correlation IDs are ordinary 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. Several application-specific properties may be set in the property section of the JMS message. Application property filters try to match these properties 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 $\#13$. Unlike correlation ID filters, a combination of different properties may be specified which leads to more complex filters with a finer granularity.

In this work we consider only application property filters, which search for so called StringProperties. Further investigations on this topic are published in [3]. We call a filter, which is searching only for one StringProperty (value), simple filter. If a filter contains logical operators, like “OR” or “AND” as concatenating elements of different components of this filter, we call it a complex filter. A complex filter searching for StringProperties is structured like the following example:

```
ID_1 = "0000" AND ID_2 = "0001" AND ... AND ID_x = "0000"
```

Corresponding to the structure of the filter the sent messages contain matching pairs of keys and values, which are set in the application property header part of a message.

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 point of view 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 same authors 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.

In previous work [3, 17–19] we already examined the message throughput performance behavior of different JMS servers, e.g. the FioranoMQ, WebsphereMQ, SunMQ, and the ActiveMQ. These investigations cover the dependency of the server performance on the number of installed publishers, subscribers, and we provided for each of the servers an analytical model to predict the message processing time based on the message replication grade and the number of installed simple filters. The current work differentiates from these studies that an analytical model for joint impact of the message replication grade and complex AND- and OR-filters is developed. Since complex filters may have different length, the experiment design is more complex and a significantly larger amount of experiments is required.

3 Test Environment

Our objective is the assessment of the message throughput of the ActiveMQ JMS server with various filter configurations. 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 "Intel PIV" 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 [20], 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 Configuration

The ActiveMQ server version 4.0 stable [21] 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 Method

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 load 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, 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, the message throughput is 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 and last seconds, where the system is not in a stable condition. 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 Impact of Filters on the Message Throughput

Our main objective is to characterize the impact of different filter types on the message throughput. We conduct suitable experiments, perform throughput measurements, propose an analytical model to capture the performed behavior, and fit its parameters based on the measurement.

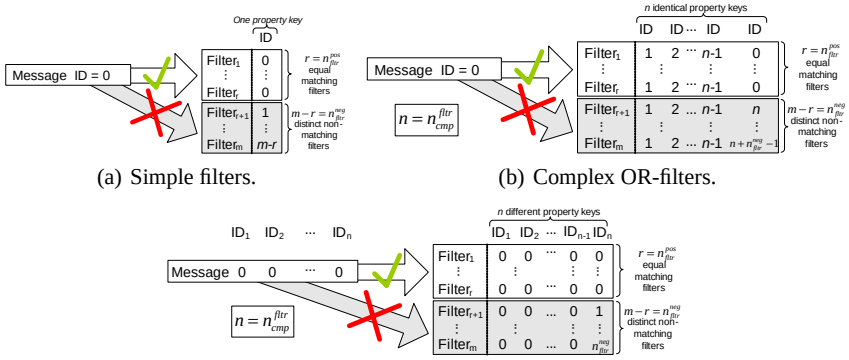
We focus on three different kind of filter types: simple filters, complex OR-filters, and complex AND-filters. For all experiments we use one dedicated ActiveMQ JMS server machine. We connect 20 publishers, distributed over two publisher machines, each of them carrying 10 publisher threads. Filters evaluate user defined message headers where we set searchable StringProperties as application properties. We use for the StringProperties a string representation of four digit numbers with potentially leading zeros. The following experiments are based on a common principle. The publishers send messages with a certain header value and n_{fltr}^{pos} subscribers filter for this value such that each message is replicated $r = n_{fltr}^{pos}$ times. The additional n_{fltr}^{neg} filters do not match, but they cause additional workload on the server. Thus, altogether $m = n_{fltr}^{pos} + n_{fltr}^{neg}$ subscribers are connected to the server and they are distributed over two subscriber machines. Each of the m subscribers maintains one exclusive TCP connection to the JMS server.

4.1 Experiment Setup

In the following, we describe the experiments for the investigation of simple filters, complex OR-, and complex AND-filters.

Simple Filters We already examined the impact of simple filters in [3] with the following experimental setup. The publishers send only messages with ID #0. As depicted in Figure 3(a), we install n_{fltr}^{pos} matching filters searching for ID value #0. Additionally we install n_{fltr}^{neg} different non-matching filters that search for values between #1 and #(n_{fltr}^{neg}).

Complex OR-Filters We consider OR-filters with n_{cmp}^{fltr} components. As illustrated in Figure 3(b), we install n_{fltr}^{pos} equal complex OR-filters, searching for ID #0 set in the last component. As the matching filter component is in the last position, no early match can save processing power when the server evaluates the filter components from left to



(a) Simple filters.

(b) Complex OR-filters.

(c) Complex AND-filters.

Fig. 3. Filter arrangements for the different experiments.

right. The publishers send messages with ID #0 to produce a message replication grade of $r = n^{pos}_{fltr}$. The last components of the n^{neg}_{fltr} non-matching filters take values from $\#(n^{fltr}_{cmp})$ to $\#(n + n^{neg}_{fltr} - 1)$ with $n = n^{fltr}_{cmp}$.

Complex AND-Filters We consider AND-filters with n^{len}_{fltr} components. The publishers send messages with value #0 for each component ID_i. As illustrated Figure 3(c), n^{pos}_{fltr} subscribers install matching filters. The values set in the last component of the n^{neg}_{fltr} non-matching filters take values between #1 and $\#n^{neg}_{fltr}$.

4.2 Results of the Measurement Experiments

We present the results for the experiments described in Section 4.1 for the parameters $n^{pos}_{fltr} \in \{1, 2, 5, 10, 20, 40\}$, $n^{neg}_{fltr} = \{1, 5, 10, 20, 40, 80, 160\}$, and $n^{fltr}_{cmp} = \{1, 2, 4, 8\}$ where applicable.

The solid lines plotted in Figures 4–6 show the measured message throughput of the ActiveMQ JMS server. The left figures present the received throughput and the right figures the overall throughput. We observe in all experimental studies a similar behavior. With an increasing number of filters the received and the overall throughput is reduced only slightly. An increasing message replication grade decreases the received message throughput, but it increases the overall message throughput. The figures for the overall throughput show a limitation of the overall throughput at approximately 50000 msgs/s. We take this observation into account when we fit the model parameters in the next section.

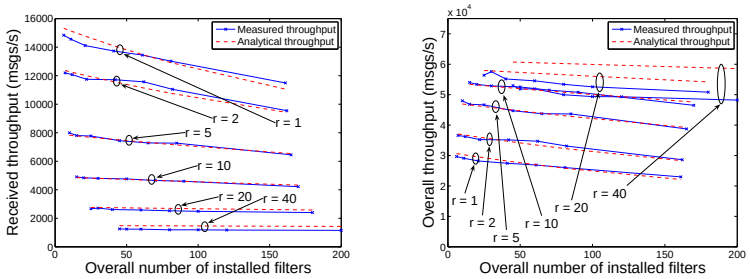


Fig. 4. Measured and analytical message throughput for simple filters depending on the message replication grade r.

5 An Analytical Model for the Message Throughput in the Presence of Complex Filters

We use the measurement results of Section 4 as input for the analytical model of the message throughput. This model improves the understanding of the server performance of the ActiveMQ as well as the impact of different parameters like the number of filters, the filter type, and the replication grade.

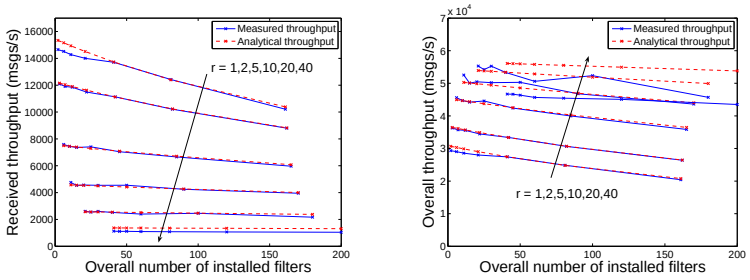
Our model assumes three different parts of the processing time for a message. Each message requires a constant overhead t_{rcv}. The processing time t_{fltr} per installed filter depends on the overall number of installed filters $m = n_{fltr}^{pos} + n_{fltr}^{neg}$ and on their length n_{fltr}^{len}. Finally, the potential replication and transmission of a message takes t_{tx} time per outgoing message. Thus, the message processing time B can by calculated by

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

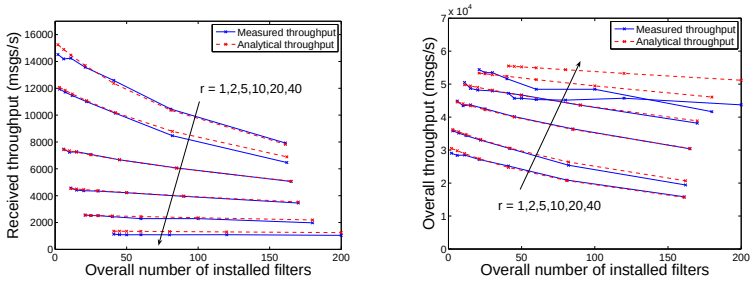
The empirical service time can be derived from the received message throughput of the measurement results in Section 4.2. The parameters t_{rcv}, t_{fltr}, and t_{tx} are fitted to the proposed model by a least square approximation. To that end we take only those curves into account that are not limited by the 50000 msgs/s margin. The parameters are derived separately for the simple, complex OR- and AND-filters. Table 1 summarizes their values. We observe that these empirical values of the model parameters are similar for all three experiment series.

Table 1. Empirical values for the parameters of the model given in Equation 1

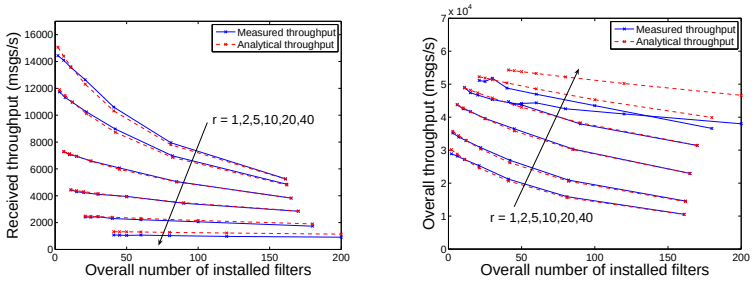
	t_{rcv}	t_{fltr}	t_{tx}
Simple filters	$4.88 \cdot 10^{-5}$ s	$1.62 \cdot 10^{-7}$ s	$1.55 \cdot 10^{-5}$ s
Complex OR-filters	$4.79 \cdot 10^{-5}$ s	$1.96 \cdot 10^{-7}$ s	$1.69 \cdot 10^{-5}$ s
Complex AND-filters	$5.19 \cdot 10^{-5}$ s	$1.86 \cdot 10^{-7}$ s	$1.71 \cdot 10^{-5}$ s



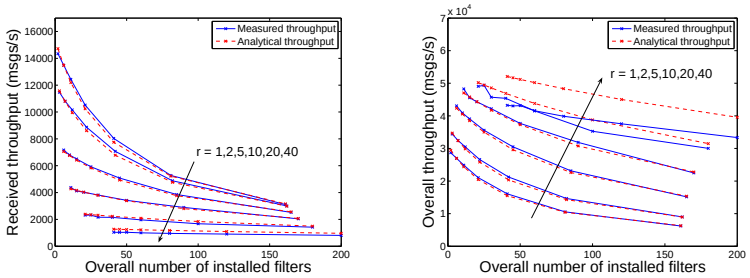
(a) Filter length $n_{cmp}^{fltr} = 1$.



(b) Filter length $n_{cmp}^{fltr} = 2$.

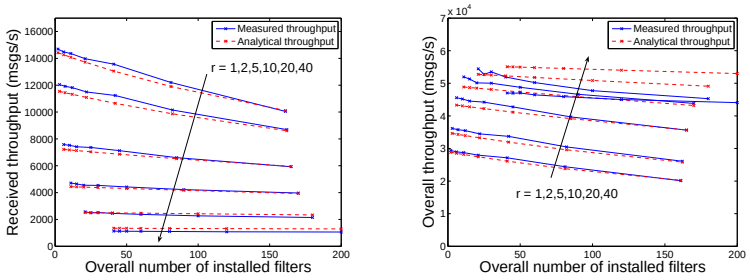


(c) Filter length $n_{cmp}^{fltr} = 4$.

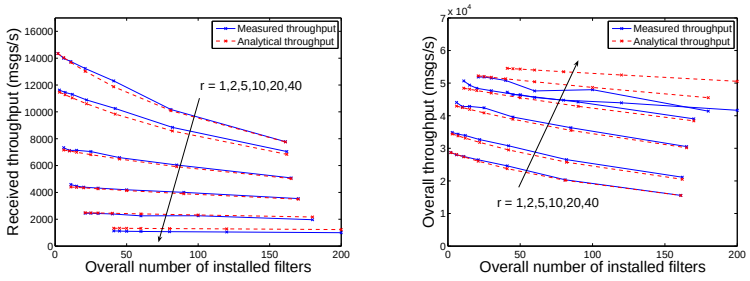


(d) Filter length $n_{cmp}^{fltr} = 8$.

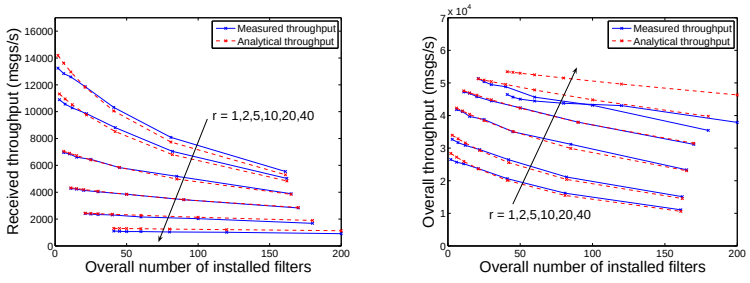
Fig. 5. Measured and analytical message throughput for complex OR-filters depending on the message replication grade r.



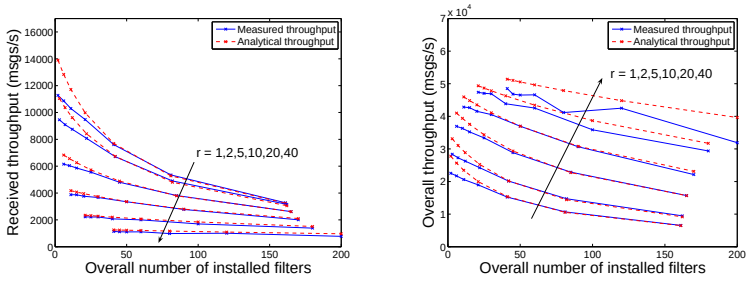
(a) Filter length $n_{cmp}^{fltr} = 1$.



(b) Filter length $n_{cmp}^{fltr} = 2$.



(c) Filter length $n_{cmp}^{fltr} = 4$.



(d) Filter length $n_{cmp}^{fltr} = 8$.

Fig. 6. Measured and analytical message throughput for complex AND-filters depending on the message replication grade r.

Based on the model and the parameters we calculcate the analytical values for the received ($\frac{1}{B}$) and overall throughput ($\frac{r+1}{B}$). They are plotted as dashed lines in Figures 4–6. For small values of the replication grade $r = \{1, 2, 5, 10\}$ the analytical data approximate the measured data very well. If the replication grade is large, i.e. $r = \{20, 40\}$, the limit of 50000 msgs/s for the overall throughput of the server is reached and the analytical model overestimates the measured throughput.

A throughput comparison of the ActiveMQ with FioranoMQ, SunMQ, and WebsphereMQ [22] shows that the ActiveMQ outperforms them by far with respect to the simple filters. Their performance is described by similar but not equal models and their time to process a simple filter is about $1.46 \cdot 10^{-5}$ s, $2.11 \cdot 10^{-5}$ s, and $1.10 \cdot 10^{-5}$ s, respectively, which explains the superiority of the ActiveMQ for use cases with extensive filtering.

6 Conclusion

In this work we have studied the impact of simple filters, complex OR-filters, and complex AND-filters on the message throughput of the ActiveMQ JMS server. The special focus of this work is on the length of OR- and AND-filters while previous work considered only simple filters with a single component. We extended an analytical model based on this previous work. The newly proposed formula for the message processing time in Equation (1) respects the number m of filters installed on the JMS server, their lengths n_{fltr}^{cmp}, and the message replication grade r. We received measurement results based on appropriately designed experiment series. These results allow us to fit the model parameters. The analytical throughput derived by the model was in good accordance with the measured results. Surprisingly, the impact of all filter types on the message processing time is almost the same and only the number of components within a filter significantly influences the time required for its evaluation.

After all, the presented model improves the understanding of general JMS server performance. In addition, the model is useful to predict the message throughput for use cases in advance when the mean values of the critical parameters are known. This obsoletes extensive hardware experimentation and makes the formula attractive for application in practice by engineers. Of course, the absolute values of the presented throughput for the ActiveMQ are only valid in our test environment. However, our presented methodology and the specially designed experiment series may be used for the performance evaluation of other environments and other server types.

References

1. Sun Microsystems, Inc.: Java Message Service API Rev. 1.1. (2002) `http://java.sun.com/products/jms/`.
2. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M.: The Many Faces of Publish/Subscribe. In: ACM Computing Surveys. (2003)
3. Henjes, R., Schlosser, D., Menth, M., Himmler, V.: Throughput Performance of the ActiveMQ JMS Server. In: ITG/GI Symposium Communication in Distributed Systems (KiVS), Bern, Switzerland (2007)

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/resources/software/enterprise_backbone/message_service.pdf.
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. (2005) http://developers.sun.com/prodtech/msgqueue/reference/docs/index.html.
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: 8^{th} 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 and Computation: 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. Henjes, R., Menth, M., Gehrsitz, S.: Throughput Performance of Java Messaging Services Using FioranoMQ. In: 13^{th}GI/ITG Conference on Measuring, Modelling and Evaluation of Computer and Communication Systems (MMB), Erlangen, Germany (2006)
18. 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)
19. Henjes, R., Menth, M., Zepfel, C.: Throughput Performance of Java Messaging Services Using WebsphereMQ. In: 5^{th}International Workshop on Distributed Event-Based Sytems (DEBS) in conjuction with ICDCS 2006, Lisbon, Portugal (2006)
20. Sun Microsystems, Inc.: JRE 1.5.0. (2006) http://java.sun.com/.
21. Apache: ActiveMQ, Reference Documentation. (2006) http://www.activemq.org.
22. Menth, M., Henjes, R., Gehrsitz, S., Zepfel, C.: Throughput Performance of Popular JMS Servers. In: ACM SIGMETRICS (short paper), Saint-Malo, France (2006)