Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Kommunikationsnetze
Prof. Dr. P. Tran-Gia

Performance Evaluation of Publish/Subscribe Middleware Architectures

Robert Henjes

Würzburger Beiträge zur
Leistungsbewertung Verteilter Systeme

Bericht 04/10

Würzburger Beiträge zur

Leistungsbewertung Verteilter Systeme

Herausgeber

Prof. Dr. P. Tran-Gia
Universität Würzburg
Institut für Informatik
Lehrstuhl für Kommunikationsnetze
Am Hubland
D-97074 Würzburg

Tel.: +49-931-31-86630
Fax.: +49-931-31-86632
email: trangia@informatik.uni-wuerzburg.de

Satz

Reproduktionsfähige Vorlage vom Autor.
Gesetzt in LaTeX Computer Modern 9pt.

ISSN 1432-8801

Performance Evaluation of Publish/Subscribe Middleware Architectures

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius–Maximilians–Universität Würzburg

vorgelegt von

Robert Henjes

aus

Würzburg

Würzburg 2010

Eingereicht am: 30.09.2010
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr.-Ing. P. Tran-Gia
2. Gutachter: Prof. Dr. Peter Sturm
Tag der mündlichen Prüfung: 23.11.2010

Danksagung

Während der letzten 9 Jahre am Lehrstuhl für Kommunikationsnetze bin ich vielen Menschen begegnet, die mich zunächst als Student und die letzten 6 Jahre dann bei meiner Forschung und der Erstellung meiner Arbeit unterstützt haben. All jenen möchte ich an dieser Stelle danken.

Meinem Doktorvater und Betreuer, Herrn Prof. Dr.-Ing. Phuoc Tran-Gia, gilt ganz besonderer Dank, da er mich von Anfang an in meiner Forschung unermüdlich unterstützt hat, nicht zuletzt durch viele Diskussionen. Auch möchte ich mich dafür bedanken, dass er mir bei allen Aktivitäten stets sein Vertrauen geschenkt hat. Mit seiner Hilfe war es mir erst möglich, meine Erfahrungen im internationalen Umfeld auf Konferenzen, bei der Kooperation mit Industrie oder in der technischen Betreuung eines Lehrstuhlnetzes zu erlangen, auszubauen und in neue Ideen für meine Arbeit umzusetzen. Nicht zuletzt durch das angenehme Arbeitsklima, das durch ihn am Lehrstuhl geschaffen wurde, wird mir die Zeit während meines Studiums und meiner Promotion in Würzburg immer in positiver Erinnerung bleiben.

Meinem Zweitgutachter Herrn Prof. Dr. Peter Sturm danke ich für seine Aufmerksamkeit, die er meiner Arbeit geschenkt hat, sowie Prof. Dr. Alexander Wolff und Prof. Dr. Reiner Kolla für das Interesse an meiner Arbeit, und dass sie als Prüfer für meine Disputation zur Verfügung standen.

In meiner Zeit am Lehrstuhl durfte ich in verschiedenen Projekten mitwirken und möchte mich deshalb bei allen mitwirkenden Projektpartnern bedanken. Besonderer Dank gilt den Mitarbeitern der Firma Siemens Enterprise Communications (SEN), die die wesentlichen Fragestellungen zu meiner Arbeit beigetragen haben. Auch den Partnern bei der DATEV und im Projekt German-Lab möchte ich für deren Einsatz und ihre Unterstützung danken.

Ein besonderer Dank gilt auch Prof. Dr. Michael Menth, durch dessen, im wahrsten Sinne, unermüdliche Unterstützung ich sehr viel im Bezug auf wissenschaftliches Arbeiten lernen konnte. Auf seine fachliche Kompetenz konnte ich mich jederzeit stützen, wodurch er einen wesentlichen Anteil zum Gelingen der Forschung und der Arbeit beigetragen hat.

Besonderer Dank gebührt der Geduld und den fruchtbaren Eingebungen meiner langjährigen Bürokollegen, Dr. Andreas Binzenhöfer, Dr. Rüdiger Martin und Dr. Simon Oechsner.

Spezieller Dank gilt auch meinen Kollegen Dr. Rastin Pries, Daniel Schlosser und Michael Duelli, nicht nur für die aufmerksame und unermüdliche Korrekturarbeit. Meinem Bruder Hans möchte ich an dieser Stelle für den abschließenden Feinschliff danken. Speziell Michael Duelli, Michael Jarschel und Daniel Schlosser möchte ich außerdem für die gute Zusammenarbeit in den letzten Jahren bei der technischen Administration des Lehrstuhls danken.

Das positive Arbeitsklima, an welchem ich teilhaben durfte, wurde erst durch den Zusammenhalt, das Vertrauen und die gegenseitige Unterstützung unter den Kollegen möglich. Dies wurde besonders deutlich beim gemeinsamen Genuss von Kaffee oder Tee in dem meinem Büro nicht fernen Etagensozialraum, der eine unerschöpfliche Quelle für Ideen, Diskussionen und sonstige Aktivitäten war und ist. Deshalb möchte ich an dieser Stelle neben den bereits Genannten allen anderen jungen und „alten“ Kollegen danken, denen ich dort regelmäßig begegnet bin und die auch sonst wesentlicher Bestandteil meines Werdegangs waren, sind und hoffentlich bleiben werden: Matthias Hartmann, Dr. Klaus Heck, Matthias Hirth, David Hock, Dr. Tobias Hoßfeld, Dr. Alexander Klein, Dominik Klein, Frank Lehrieder, Dr. Kenji Leibnitz, Dr. Andreas Mäder, Dr. Rüdiger Martin, Dr. Jens Milbrandt, Prof. Dr. Oliver Rose, Christian Schwartz, Barbara Staehle, Dr. Dirk Staehle, Prof. Dr. Kurt Tutschku, Dr. Norbert Vicari, Florian Wamser und Thomas Zinner. Bedanken möchte ich mich auch im weiteren Umfeld bei Markus Sauer, Stefan Selbach und Florian Zeiger für die Unterstützung und die gemeinsamen Aktivitäten.

Viele Teile der Arbeit wurden erst durch die tatkräftige Mithilfe meiner Studenten, Praktikanten und Diplomanden möglich, sei es durch direkte Arbeit am Thema oder durch die Unterstützung bei der Ausführung meiner zusätzlichen Aufgaben. Besonders möchte ich meinen Diplomanden Sebastian Gehrsitz, Valentin Himmler, Matthias Koller und Christian Zepfel danken. Für die Unterstützung nicht nur in technischen Belangen, wie der Betreuung der IT am Lehrstuhl, möchte ich besonders den studentischen Hilfskräften Steffen Gebert, Inanc Gürültücü, Nicholas Gray, Waldemar Huber, Christopher Metter, Sven Scheuring, Stephan Seufert, Christian Sieber, Johannes Wacker und Tobias Ziermann danken, sowie allen anderen, die mir bei der Betreuung von Vorlesungen und Projekten geholfen haben.

Bedanken möchte ich mich auch bei unserer Sekretärin Frau Gisela Förster, für die Unterstützung in allen organisatorischen und verwaltungstechnischen Angelegenheiten, auch unter Zeitdruck. Des Weiteren bedanke ich mich bei Fritz Kleemann und Markus Krieger für die Unterstützung in allen technischen Angelegenheiten außerhalb unseres Lehrstuhls.

Meiner Familie möchte ich danken für den Rückhalt und die immerwährende Unterstützung in allen Lebenslagen.

Contents

1 Introduction

Modern communication mechanisms are a key element to reduce physical distances between people and enable efficient business processes. A large variety of applications and devices evolved during the last decade to support people's communication. To design an efficient information exchange in this heterogeneous scenery, the communication interfaces have to provide flexibility and performance.

Messaging introduces the possibility of synchronous and asynchronous communication while separating the communication partners by message-oriented middleware. This set of features can be illustrated best by comparing two well-known communication applications: a classic telephony system and an e-mail infrastructure. Considering the telephony system, a participant can only communicate synchronously with a single other participant if both parties are available for communication at the time the phone call is initiated. Also both sides have to speak a common language and there is no way to filter out redundant information, beside rejecting the caller. Within an e-mail infrastructure, information can be sent using messages to a central entity. The information will reside on the central entity until a receiver fetches it which enables asynchronous communication. Additionally, the receiver can decide if and in which order to consume the incoming information.

In the context of message-oriented middleware, communication is mostly asynchronous. The communication process is realized as a set of messages, rather than a continuous information flow, like observed in classic telephony. According to Hohpe and Woolf [121], messaging systems enable a "high-speed, asynchronous, application-to-application communication with reliable delivery" based on the exchange of messages over a packet-switched network.

According to the aforementioned description of the basic system architecture, we can identify six crucial components. First, the *message*, containing and transmitting the information itself. The message has to carry the plain information generated by the clients and some meta-information for the second element, the *message-oriented middleware*. The third important component is the definition of *standard communication interfaces*, e.g., the *Java message service* (JMS) [129]. A fourth component is the *deployment* where a messaging system is used in, like a data center or worldwide communication through the Internet. In addition, the underlying communication pattern influences the deployment scenario, like using a publish/subscribe-oriented communication. Exchanging information over the Internet means to transmit data through an unsecure and unreliable medium. This leads to the last two elements, the introduction of *reliability* and *security* into the messaging environment.

Since messaging, if used in application-to-application communication, represents the backbone of the system, it has to have a high, well-known performance. Therefore, it is necessary to develop mechanisms and methodologies to determine the performance limits of such systems. It is important to identify possible bottlenecks and parameters causing them. Also during deployment, a careful dimensioning of the system should take place, which can be supported by efficient prediction models adaptable to a varying set of application scenarios.

1.1 Scientific Contribution

The main contribution of this thesis is an approach to analyze and evaluate the performance of a message-oriented middleware in the context of JMS. The analysis and evaluation is based on system-level measurements in combination with an easy-to-handle model for predicting the overall system performance. The presented approach can be adapted to a wide range of application scenarios.

On the market, a huge number of messaging solutions and commercial products are available and their feature sets are diverse. Thus, the first contribution of this work is to discuss the different design options and to categorize them according to their relevance. A focus is set on the publish/subscribe communication pattern, which is supported by the JMS. However, we can assume that the messaging system in general may become a bottleneck in common application scenarios. Therefore, the overall achievable message throughput is identified as a measure for comparison of the design options and is additionally useful in different optimization scenarios. In a publish/subscribe-oriented communication infrastructure, the filtering and replicating information that are carried by a message are crucial for achieving a high message throughput of the overall system. Both aspects – filtering and replication – are a major focus in the remainder of this monograph. For evaluation, we select a significant set of servers in order to compare different implementation strategies.

To be able to determine the limits of the different servers, we introduce a black-box oriented measurement approach, which allows to test a server according to different parameters in a semi-automated environment. This approach differs from basic benchmarking approaches by its adaptability to varying application scenarios and a more fine-grained output of the results. This enables a detailed analysis of possible performance bottlenecks. Furthermore, we introduce a calibration methodology and automated measurement verification for our testbed and our experiments. This ensures the reproducibility of the measurement runs and consistent measurement results. In total, we measured different systems regarding the impact of the number of connected clients, different information filtering options, and network specific influences. We identify basic values of typical parameters to operate our tests in a significant environment. As a result, we observe that the replication grade and filtering has a high impact on the overall system performance.

Thus, we introduce a model for the JMS server's message throughput performance considering the impact of replication grade and filtering. To get the system-specific values for the prediction models, we design a dedicated experiment series and repeat them for all considered servers. The model itself is based on linear regression adapted to the requirements of our system level measurements. We further show the adaptability of our approach by enhancing the models for complex filtering scenarios and evaluate an application scenario using our models.

The server models are based on average values for the message throughput and do not consider internal server behavior, like message waiting times. To show the influence of the server behavior on the overall system we use an $M/GI/1-\infty$ queuing system for approximating the waiting times. One application scenario is the real-time prediction of the impact of a certain configuration on the overall system performance. Since there is no algorithm available with the desired efficiency for solving an $M/GI/1$–∞ queuing system numerically, we introduce a fast approximation method to calculate the required numbers and denote it by "Gamma-approximation".

In the first part of the thesis we focus on a single server scenario, whereas we discuss in the second part two different design options regarding message throughput scalability. To evaluate the expected performance of the two different approaches, we use our basic message throughput models and our findings regarding the waiting time analysis.

1.2 Outline of the Thesis

Figure 1.1 depicts the overall organization of the remainder of this monograph. The figure is organized such that each box represents a chapter of the thesis and outlines the covered topics. In Figure 1.1, all contributions resulting from this thesis are classified into the main topics of the thesis and plotted on the right side of each chapter overview. Some of the publications are cited multiple times since they cover the evaluation of a single server.

Chapter 2 introduces the publish/subscribe communication pattern. Section 2.2 discusses the widely used JMS and its features regarding the support of publish/subscribe communication. Furthermore, in Section 2.3 we present an application scenario as a motivation for our performance evaluation. The chapter also contains in Section 2.4 a discussion on related literature regarding the basic architecture and design options for publish/subscribe systems, general benchmarking of message-oriented middleware, and presents a global overview of related messaging approaches.

Chapter 3 proposes an architecture for an experimental environment in order to evaluate the performance of publish/subscribe-based systems. Moreover, it describes our measurement methodology and discusses different parameters considered for testing. In Section 3.2, we evolve a basic series of experiments in order to determine the boundaries of our equipment and the server software used. In Section 3.3 and Section 3.4, we extend our experiments regarding the impact of filtering and client handling. We conclude this chapter by analyzing the impact of different parameters tested by the measurements.

Chapter 4 introduces a general methodology to build basic models for approximating the message throughput capacity of Java message service servers. As a tool for this methodology, we present some basics on multiple regression and least-squares approximation in Section 4.1. This is followed by Section 4.2 and Section 4.3 introducing a basic version of the model regarding the joint impact of number of filters and replication grade for different server types. In Section 4.4 we extend the basic model considering complex filtering. We conclude this chapter with an application scenario of the models and some remarks on the results of our proposed models.

Chapter 5 focuses on the message waiting time of a messaging server and an $M/GI/1-\infty$ queuing system is applied. Since the focus is on real-time numerical evaluation of the queuing behavior, we introduce an approximation method based on the Gamma-distribution in Section 5.1. The analytical performance evaluation considering the Gamma-approximation method is presented in Section 5.2. In addition, we discuss two different options to introduce scalability regarding the overall message throughput in Section 5.3.

Chapter 6 summarizes the major contributions of this work. Additionally, an outlook on future trends and challenges in the area of publish/subscribe systems and messaging systems in general is given.

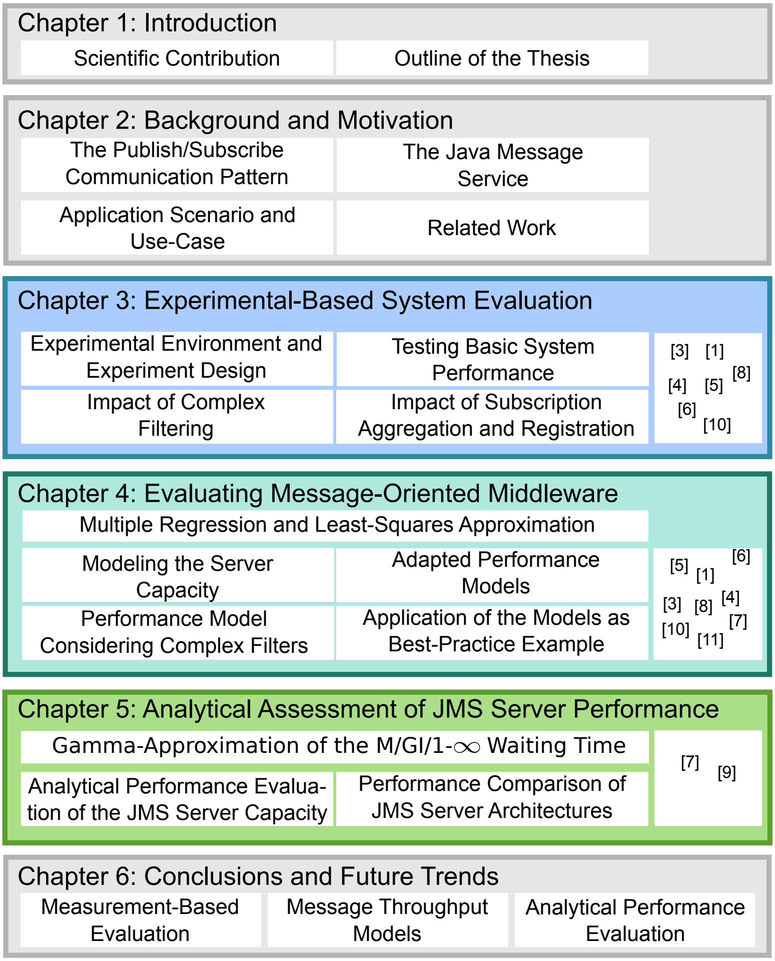

Chapter 1: Introduction
Scientific Contribution
Outline of the Thesis
Chapter 2: Background and Motivation
The Publish/Subscribe Communication Pattern
The Java Message Service
Application Scenario and Use-Case
Related Work
Chapter 3: Experimental-Based System Evaluation
Experimental Environment and Experiment Design
Testing Basic System Performance
Impact of Complex Filtering
Impact of Subscription Aggregation and Registration
[3] [1] [8] [4] [5] [6] [10]
Chapter 4: Evaluating Message-Oriented Middleware
Multiple Regression and Least-Squares Approximation
Modeling the Server Capacity
Adapted Performance Models
Performance Model Considering Complex Filters
Application of the Models as Best-Practice Example
[5] [1] [6] [3] [8] [4] [10] [11] [7]
Chapter 5: Analytical Assessment of JMS Server Performance
Gamma-Approximation of the M/GI/1-∞ Waiting Time
Analytical Performance Evaluation of the JMS Server Capacity
Performance Comparison of JMS Server Architectures
[7] [9]
Chapter 6: Conclusions and Future Trends
Measurement-Based Evaluation
Message Throughput Models
Analytical Performance Evaluation

Figure 1.1: *Organization and contribution of the thesis.*

2 Background and Motivation

In this chapter, basic principles of *message-oriented middleware* (MOM) are presented and discussed. The communication pattern “publish/subscribe” and its industry standard framework called *Java message service* (JMS) are introduced. We discuss a set of parameters which influence the performance of a JMS environment. Furthermore, an overview is given on related work in the areas of publish/subscribe in general, JMS framework implementations, and performance evaluations of message-oriented middleware.

2.1 The Publish/Subscribe Communication Pattern

In general, message-oriented middleware (MOM) acts as a mediation platform for the communication of application components and allows them to create, send, and receive messages. Acting as a mediation platform, MOM provides distributed communication which is loosely coupled, reliable, and asynchronous. The involved entities can be divided into a message producer, a message consumer, and the mediation entity. The mediation entity manages one or multiple queues, which are used to coordinate the message transmission process. The entities can be distributed on multiple machines or run on a single machine. The process of information dissemination is directed from the producer to the consumer. Overall we can distinguish a *point-to-point* (PtP) and a *publish/subscribe* (pub/sub) oriented messaging approach.

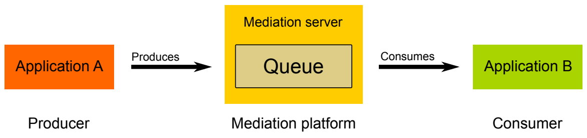


Figure 2.1: *Point-to-Point messaging pattern.*

2.1.1 The Basic Communication Patterns using MOM

The PtP messaging approach defines the communication of a message producer with a single consumer as depicted in Figure 2.1. The producer and consumer are connected using a server application which provides a dedicated queue for the communicating entities. A message producer sends its messages to the desired queue. In case multiple consumers are connected to the same queue, the PtP pattern ensures that a message is delivered only once to a single consumer. It is not necessary that the producer and the consumers are connected to the server at the same time as described in [129]. A message will be retained in a queue until it is fetched by a single consumer or will be dropped as soon as its expiration date is reached. In order to support multiple applications an individual queue has to be set up on the server for each pair of communicating applications.

In contrast to PtP, the pub/sub communication pattern provides a one-to-many communication. As Figure 2.2 illustrates, multiple consumers subscribe to the same topic. An application publishes a message to a central topic located on a mediation platform, which is very similar to the PtP process. The messages will be dispatched to each subscribed application on the consumer side. The mediation platform itself might be centralized or distributed among different servers. In general, the mediation platform divides the communication entities in publishers and subscribers, which is also called decoupling. Eugster et al. [57] differentiate between a full decoupling in space, synchronization, and time in pub/sub-based systems.

Considering *space decoupling* publishers and subscribers do not communicate directly with each other, but use an event router as mediation platform. Therefore,

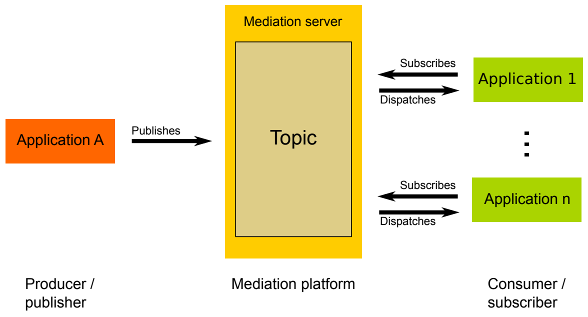


Figure 2.2: *Publish/Subscribe communication pattern.*

the publishers do not have to know the subscribers and their locations. On the other hand the subscriber does not need to know who published the message. This guarantees a kind of anonymity between the communication partners.

The publish/subscribe system further establishes *synchronization decoupling*. The one-way character of the message transmission provides an asynchronous communication between publishers and subscriber. Hence, publishers are not blocked while they produce messages and subscribers are notified asynchronously of a message. In the meantime they can perform some concurrent task.

In case of *time decoupling* the mediation platform offers a message queue which enables an asynchronous communication, where publishers and subscribers have not to be available at the same time. The published messages e.g., are delivered at a later point in time in case of a network failure.

2.1.2 Variations of Pub/Sub Systems

One of the most crucial aspects in messaging systems is to avoid flooding of the consumers with unwanted messages. Typically a consumer is only interested in a subset of all published messages. The pub/sub communication pattern tries to address this problem by offering options for respecting the consumer interests.

A very coarse grained pub/sub scheme is based on so-called *topics*. A topic divides the overall message load into logical subclasses. A publisher sends its messages to the corresponding topic, and each subscriber interested in that particular topic will receive the message. Topic-based systems are also often referred to as subject-based systems. Topics can be easily distributed over multiple servers, since they are independent. This kind of organization enables a straightforward scalability with an increasing message load. A major disadvantage of topic-based systems is that the topics have to be defined previously on an administrative level. So the scalability of a topic-based system is limited by the capacity of a server hosting the topic with the highest load.

However, *content-based* pub/sub provides mechanisms to introduce message filters by the subscribers to specify their interests. One of the early goals of the content-based pub/sub approach is to match the interests of a subscriber against the published information on a semantic level. Seen from a technical level the mediation platform has to consider the whole content of a message. This is a very time-consuming task which does not scale regarding the overall message throughput.

Therefore the message is divided into a so-called *application header* and a body part containing the information payload. The application header stores attributes and tags describing the properties or content of a message. The values of the attributes and tags are searchable on the mediation platform by a varying set of filtering languages. This kind of routing decision is also called *header-based* pub/sub system. In environments where the mediation servers are distributed, the filters have to be propagated throughout the system. Since the filter propagation is a time-dependent process, the state of the different servers might become inconsistent. This might lead to unexpected behavior while disseminating messages. Problems regarding distributed event routing are not part of this work, but are well-known in literature, e.g. [87].

Several additional approaches try to optimize the performance of a central mediation platform. E.g., additionally to the propagation of the subscriber side interests a publisher can previously advertise its attributes and a corresponding value range, which enables additional possibilities to improve the internal mediation server performance, as shown in [137].

2.1.3 Performance Characterization in Pub/Sub

Overall two characteristics are important: (1) the expressiveness of filtering and (2) the achievable maximum message throughput of the mediation platform. The *expressiveness* determines the granularity at which the interests of the subscribers can be captured. For example in header-based pub/sub systems the number of attributes introduced to each message, respectively the length of a filter are influenced by the expressiveness. The *maximum message throughput* of a mediation platform is a key performance parameter, since it might represent the overall performance bottleneck, due to the central character of the pub/sub system.

Furthermore, other requirements which have to be considered in an event routing environment are the timely delivery of messages, reliability of the service, as well as preserving the message transmission ordering. Also guarantees for message delivery and roll-back mechanisms in case of a failure are key features in industry standard implementations.

2.2 The Java Message Service

The *Java message service* (JMS) is an *application programming interface* (API) provided by Sun Microsystems (now Oracle) as described in [129] and [120]. It is organized as a programming framework for Java, but considers a generic system behavior. In general, the JMS framework defines the system as a set of non-implemented Java methods such as interfaces and abstract methods, where the specific implementations are up to a vendor.

The API defines Java interfaces for the publishers how to generate and send messages to the JMS server. For the subscriber side, the defined Java interfaces consider the reception of these messages – or a subset thereof – from the JMS server. The API provides abstract Java methods to control the message flow by various message filtering options. The JMS server itself, which represents the mediation server, is not specified by the API and its implementation is up

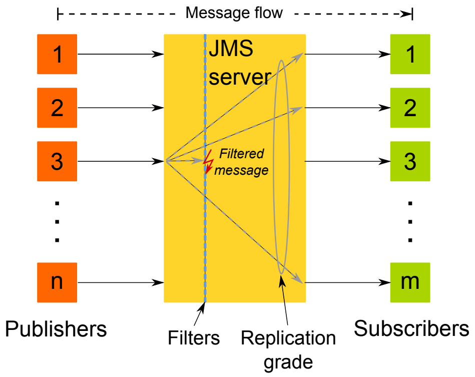


Figure 2.3: *The JMS server defines the participating entities as well as the message structure.*

to a vendor, as well as the underlying communication mechanisms between publishers and subscribers. In this context the replication grade describes the number of messages which have to be transmitted to the subscribers by the JMS server for a single published message. The overall architecture of the JMS framework is depicted in Figure 2.3. In this section, we describe the JMS framework and the most important features provided by the JMS API.

2.2.1 JMS Message

One of the basic components of the JMS API is the definition of interfaces to represent a message. In JMS context, a message is split into an application header part and a message body, as depicted in Figure 2.4. The application header consists of a fixed and an application-specific part, while the body contains solely payload data.

The JMS API defines up to 10 different fields in the *fixed part* of the header. A detailed description can be found in [141]. We introduce briefly the most common parameters as outlined in Table 2.1. When publishing a message, some

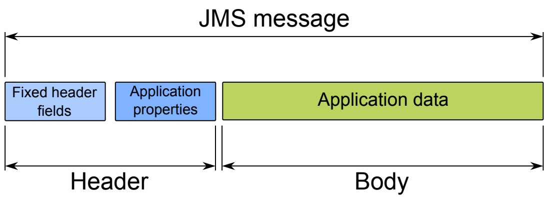


Figure 2.4: *Structure of a JMS message.*

of the properties can be set by the publisher application, while others can only be set by the vendor specific publisher implementation or the JMS server itself. Upon reception the subscriber application can override all properties with custom values.

Table 2.1: *Selected properties of the fixed header part in the JMS API.*

Field name	Value is set by	Provided by	Optional
JMSDestination	Publisher	Vendor	No
JMSDeliveryMode	Publisher	JMS API	No
JMSMessageID	Vendor/Publisher	Vendor	No
JMSCorrelationID	Publisher	Client	Yes
JMSRedelivered	JMS server	JMS API	No

The *JMSDestination* parameter contains the destination of a message, e.g., the name of a topic or a queue. The value must remain constant until reception of the message by a subscriber. The *JMSDeliveryMode* controls, if a possible loss of the transmitted message is tolerable. If it is set to persistent, a subscriber has to receive successfully the message “once and only once”, as stated in the JMS API. Delivering a message twice may cause undefined behavior, for example if the message contains only a differential update to a state stored at the subscriber. E.g., consider a financial balance which is updated twice by a positive value. In contrast, non-persistent defines only a reception by at most once and message loss is not a concern. The *JMSMessageID* and the *JMSCorrelationID* are two parameters for message identification. The message ID is set by the vendor specific publisher implementation, which cannot be influenced by the customer application. However, the correlation ID can be set by the customer application to mark a sequence of messages. Especially it is possible to search this parameter by a subscriber's filter. The *JMSRedelivered* field is an indicator that is set by the JMS server in case a message has to be transmitted twice, e.g., due to a message loss on the network path. This option holds for persistent and non-persistent mode and marks, if a message is delivered a second time, due to a possible loss of a JMS message acknowledgement.

The JMS API also defines who provides the possible values for the properties. Some of the properties have defined a fixed value range provided by the JMS API, like the *JMSDeliveryMode* and the *JMSRedelivered* parameter. Other properties, like the *JMSDestination* are specific implementations provided by the vendor of the JMS server. Some of the parameters can be customized by the client application according to its demands, like the *JMSCorrelationID*.

In contrast to the fixed header part, several *application-specific properties* may be set in the application property section of a JMS message. Application properties can be specified by the application where the name of the property and its value are free to be adapted to the needs of an application designer. They can store common data types, like *booleans*, *bytes*, *integers*, *floats* and *strings*. The overall number of arguments is not limited for a message.

2.2.2 Message Selection (Filtering)

In contrast to the mechanisms on publisher side, a subscriber has the ability to select messages according to different header properties. In the JMS framework, pub/sub is limited to the header-based routing concept which considers only properties in the header of the JMS message and does not allow to select messages upon the content of their message body. This limitation is acceptable for performance and scalability reasons. The JMS framework defines a subset of the SQL92 [22] conditional expression syntax as selection language which enables an increased complexity in designing message selectors. In this work we focus on the operands presented in Table 2.2.

Table 2.2: *JMS SQL92: Important message selection operands.*

Operand	Data type
$=$	All
$<, \leq, >, \geq, \neq$	Arithmetic
`AND`, `OR`, `NOT`	Boolean
`BETWEEN`	Arithmetic
`IN`	String
`LIKE`	String

The JMS framework does not specify which entity has to process the filter. But most vendors evaluate the installed selectors on the JMS server for obvious reasons. The JMS server will only forward messages to subscribers if their installed SQL92 statement evaluates to `TRUE` compared with the values set in the properties of a message. The JMS framework specifies an operand `LIKE` that allows a message selection based on wild-cards in context of string based data types, which is not considered in our evaluations. The impact on filter evaluation time by specifying `LIKE`-based fuzzy searches does not apply to the high performance requirements of a data center application scenario.

2.2.3 Message Transmission Modes and Reliability

In general, the JMS server does not define any mechanisms or methods to provide a guaranteed reliability and availability of the JMS service itself. The JMS framework requires only a specific behavior for reliable message delivery by defining constraints. The implementation and the methods used to keep these constraints are up to the vendor of the JMS server environment.

2.2.3.1 Differentiation of Service and Data Availability

Since the development of the JMS framework has been mainly driven by Sun Microsystems, it is worth to have a look on their implementation strategies within the *Sun Java System Application Server* [138], now known as the

Oracle System Application Server. The application-specific documentation of this server introduces two levels of availability, the *service availability* and the *data availability*. With respect to the pub/sub communication pattern this differentiation can be considered as generally admitted.

With *service availability*, it has to be ensured that the JMS service continues its operation with a minimized down-time in case of a failure by adding sufficient redundancy. Single messages might be lost as long as the JMS service resumes operation with only a minimal delay. Most common architectures cope with this problem by providing standby backup servers or a cluster of active servers.

Data availability means the persistent and consistent information and message handling. This availability level enables the guarantees as defined by the JMS API, the once and only once message delivery (persistency) and the message ordering (consistency). Data availability requires increased overhead compared to pure service availability. Service availability is out of scope in this work, whereas data availability is partly focused by evaluating the connection times of a large number of subscribers, e.g., after a system failure.

2.2.3.2 JMS Acknowledgement Modes

One important feature defined by the JMS API to ensure valid message transmission is to acknowledge the reception and transmission of messages on their way through the mediation environment. This enables to delegate responsibilities to subsequent entities if they acknowledge correctness on application level. In general, JMS does not consider a message sent successfully until the subscriber acknowledges the reception of the message. The mediating entities store a transmitting message until the appropriate acknowledgment arrives. The JMS framework describes three different mutually exclusive acknowledgment modes.

If the `DUPS_OK_ACKNOWLEDGE` mode is active, the JMS implementation acknowledges only “lazily” the reception of messages. However, the JMS specification does not state precisely what “lazily” in this context exactly means. But by activating this option the JMS server can benefit from a reduced overhead while maintaining its resources for persistent message delivery. In this context the subscriber implementation must be tolerant to the reception of duplicate messages.

By default the mode `AUTO_ACKNOWLEDGE` is active. In this mode, the vendor specific JMS implementation is responsible for sending acknowledgments upon reception of a message. The subscriber application does not have to pay attention to acknowledge incoming messages. Since this mode internally follows the once-and-only once policy, it introduces overhead on the JMS server itself.

Activating the third mode `CLIENT_ACKNOWLEDGE` a subscriber application itself becomes responsible acknowledging a message reception. The underlying JMS environment should be aware to limit the number of unacknowledged messages. This protects a subscriber from message overload.

2.2.3.3 JMS Message Delivery Modes

The JMS offers several modes to ensure persistent message delivery. Activating the *persistent* mode, messages are delivered reliably and in order. This option can be set by each individual publisher and affects all its published messages. In addition, a publisher can decide to set this option on a per message basis, if its default is set to *non-persistent*. In both cases, a JMS server must not deliver a message twice, especially in case of an outage. If the persistent mode is activated the JMS server has to store a copy of this message, until the message is successfully delivered to all interested and currently connected subscribers. This might lead to a delayed delivery and therefore introduces an increased overhead on the mediating entity.

In addition to the persistence options for messages, which are commonly set by the publisher, a subscriber can define the mode to receive messages. In the *durable* mode, messages will be also forwarded to previously registered 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 and it might achieve a larger throughput in the non-durable mode.

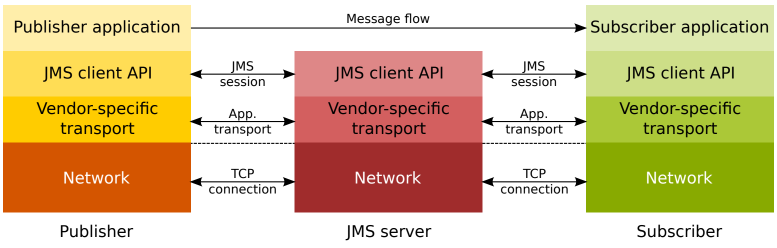


Figure 2.5: *Communication layer in the JMS application environment.*

Similar to databases the JMS framework also offers so-called *transacted* sessions. Here, in one transaction a set of messages is treated as a group. The transaction itself is an atomic unit of work. In case of a failure, the complete set of messages within one transaction has to be destroyed. This is also called *roll-back*. So either all messages arrived successfully at all interested subscribers, or none of them.

2.2.4 Network Level Communication and Application-Layer Transport

JMS introduces application-specific communication layers for connections in addition to the ISO/OSI layer model. The use of the network and transport layer is not explicitly defined by the JMS API. Typically, a TCP/IP connection is considered to be used, as depicted in Figure 2.5. Most vendors introduce additional application-layer transport protocols, like the *advanced message queuing protocol* (AMQP) [97], as an abstraction layer between transport layer and an application-specific JMS session. These transport protocols are typically adapted to the environment and the requirements of the JMS application, e.g., reliability, performance, or compatibility.

To simplify the architecture and reduce the overhead per client, it is possible to group several JMS sessions into one transport or TCP connection, respectively.

This is also used in the test environment for emulating multiple clients on a single machine, as long as the performance characteristics of different clients or TCP connections can be neglected.

2.3 Application Scenario and Use-Case

This section introduces an application scenario for the publish/subscribe communication pattern based on a JMS environment. This scenario is used as a motivation for our message-oriented middleware (MOM) evaluations presented in this work.

2.3.1 MOM Application Deployment Scenarios

In general, we can distinguish two major deployment scenarios for a pub/sub-based system. The first scenario considers an *Internet-scale* deployment of the event routing engines, as often described in pub/sub related literature, e.g., in [46] and [49]. The second scenario concentrates on enabling an event based infrastructure for a *service oriented architecture* (SOA) [115] in a local data center, e.g., as applied in [89].

Considering the Internet-scale scenario, a large number of cascaded event routers are involved for transmitting messages and maintaining the pub/sub infrastructure. Additionally, an efficient routing engine has to be optimized for low bandwidth consumption and delay tolerant networks. Also the available resources on an event router node are typically limited. Often a self-organizing approach for maintaining the topology and the subscriptions is used.

In contrast, a data center offering a service-oriented application maintains a local infrastructure with careful dimensioned server environments and network connections. As depicted in Figure 2.6, this kind of scenario can be divided into three parts according to [121]: (1) the *presentation layer* representing the customers, (2) the *business logic layer* where all service application logic is located, and (3) a *data layer* for supporting high-performance data storage.

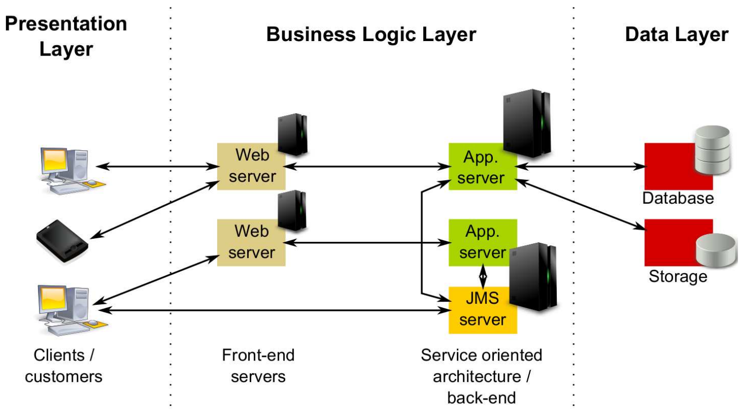


Figure 2.6: *Data center scenario implementing pub/sub.*

The presentation layer is typically implemented by a web browser or a small dedicated application, also called clients. The clients are normally connected via the Internet to the business logic of the data center and might therefore experience best effort network conditions. Additionally, the clients might be equipped with less system resources to reduce price of the equipment or the power consumption in case of mobile devices.

In the business logic layer, we can differentiate two different sub-layers, i.e., the front-end and the back-end area. The front-end is responsible for abstracting the application logic from content delivery. This approach enables the scalability of the number of customers and enables additional services, e.g., optimized graphical output of the content for the customer. The back-end handles the application logic split into small services. The network connectivity within the back-end is based on a high-speed local area network (LAN), which enables a distribution of the services among different application servers, as depicted in Figure 2.6. To enable communication between the services a MOM, like a JMS environment, is introduced. This JMS environment has to handle all messages exchanged by the application logic which represents the publishers and subscribers for the transmitted information. This leads to the conclusion that the performance of the JMS environment has to be well dimensioned in order to avoid bottlenecks. We assume, that the redundancy required for resilience is part of the JMS environment and the overall data center design itself and do not consider it in this work.

2.3.2 Presence Information Exchange as a Use-Case

A typical scenario in nowadays communication infrastructure is the exchange of *presence information* as described in [37] and [35]. Presence denotes the state of software component or an individual person, e.g., the person is currently online and available for chatting. This kind of information is typically exchanged in social networks and instant messaging environments. Presence information can be used as a trigger for a specific action if the person has set a certain presence state. For example, if the user sets the state “not available for voice communication”, the calling person might be adviced to leave a text message.

The change of presence information is typically represented as an event which is published as a message from an application service acting as a publisher to the JMS environment. All interested subscribers, which can be other services and persons, have to be notified. The subscribers provide their interests typically by a set of complex filters, e.g., filtering for the desired person identifiers. The filter design can vary from one complex filter to multiple simple filters per client. Additionally, the reception of a state change might trigger a cascade of additional events in the system.

Presence is only one example for a use-case of the pub/sub communication pattern. The data center-oriented application deployment and the presence use-case motivated the evaluations presented in this work. Other possible use-cases, alternative application deployment scenarios, and approaches for proper system design are described in the following section.

2.4 Related Work

The field of content-based routing, especially the publish/subscribe communication pattern is a well-known research topic in literature. The ongoing research covers a wide range of different issues regarding the communication pattern itself, the routing and subscription optimization, the architectural design options, as well as performance evaluation. Also the topics of security, reliability, and the introduction of the communication pattern on different network layers are hot topics in the field of research, but out of scope for this work. In the following subsections, we present a generalized overview on work in the field of the publish/subscribe communication pattern and different approaches to evaluate the performance, also regarding the JMS framework.

In literature some of the terms introduced in the previous sections have additional synonyms. The mediation server or JMS server is also known as event router or event broker. In pub/sub systems the terms *dispatching*, *notification*, and *receiving* are often used as synonyms. Also *producer* and *publisher*, as well as *consumer* and *subscriber* are replaceable terms for the same entities.

2.4.1 Publish/Subscribe Architecture and Design

The acceptance of the pub/sub communication pattern on a larger scale is enabled by an increasing performance of message-oriented middleware systems and the underlying hardware. A general introduction to the architecture of pub/sub systems is given by [57] whose authors describe the basic principles of the involved entities and features of decoupling communication partners. Typical scenarios and industry-wide accepted use-cases for designing messaging solutions are described by Hohpe and Woolf [121] and Terry and Shawn [141].

Design and Architecture

Already in the early nineties, Bernstein [25] compared different middleware components for distributed system services. Its major goal is to cope with the heterogeneity of distributed computing problems using communication on lower network layers. In nowadays service-oriented architectures, the same problems like scalability and throughput might be observed, but they moved to application-layer.

Antollini et al. [83] describe the requirements for implementing a high level publish/subscribe architecture in an enterprise grade information system. In general, a definition of all entities of a pub/sub system are given as programming interfaces and an approach to integrate JMS is presented. The described mechanisms and interfaces are useful to identify realistic scenarios in the field of pub/sub systems, and we therefore considered them within our experiment design.

The design of large-scaled content-based routing is described in the theses of Mühl [49] and of Tarkoma [87]. The work of Tarkoma focuses on the event routing process itself and also considers aspects of mobility. The thesis of Mühl presents an evaluation of the scalability of the event routing infrastructure by implementing a prototype. Both works inspired the design and the selection of certain scenarios for the experiments done in this work.

Baldoni et al. [53] give an overview on the evolution of pub/sub communication systems. The authors review general issues in pub/sub systems, such as anonymity of the participants and decoupling in time and flow. The advantages and the difference between topic-based and content-based systems are discussed. Also several research topics, still up-to-date, are mentioned regarding the efficient subscription routing and fault tolerance of the servers.

Filtering Strategies

Optimizing the filter evaluation performance is one of the key issues to achieve a high message throughput. Several publications deal with the optimization of filters. Aguilera et al. [33] proposed some basic mechanisms to match the content of messages by appropriate filters. They also validated their methods by simulative studies. In [39], a filter transformation to binary decision diagram is proposed. The authors of [48] take advantage of similarities in filters installed by different subscribers. From our observations, not all servers use these options to increase the throughput by adding such optimizations to their system.

A general discussion of filter matching algorithms can be found in [81]. According to the authors, tree-based algorithms are the most efficient way to solve the filter matching problem. A generalized proof for finding all matching filters in sub-linear time is also presented in [33]. It is expected that the time to match a random event is not greater than $O(N^{1-\lambda})$, where N is the number of subscriptions.

In [41] an approach is suggested to implement high-efficient filtering algorithms. The authors claim to support 6 million subscriptions and about 600 events per second. They consider also a high rate for subscribing and un-subscribing clients. The proposed memory and CPU optimized algorithms are evaluated by measuring a prototype. The results gained from the experiments are quite promising, but the assumed scenario differs from our observations in industrial environments. “Very fast”, as indicated in the title of [41], publish/subscribe systems have to support message throughput rates orders of magnitude higher than 600 messages per second, especially in case the system has to support up to 6 million of parallel subscriptions. Typically in data center-oriented applications, subscriptions are aggregated on a per machine or per service block level, which reduces the number of parallel subscriptions in the publish/subscribe system.

Gorton et al. [58] introduce multiple mechanisms to optimize filtering in large data streams. The focus of the work considers JMS and also some tests with the JBoss JMS server are conducted. We tried to evaluate the JBoss JMS server as well, but the versions we used could not handle the load offered by our test scenarios, so we were not able to apply our evaluation method. However, the basic design goals of the tests are in line with this work.

The authors of [36] introduce an approach using B-trees, based on a *distributed data structure* (DDS) in a grid based storage network. Different easily scalable services write concurrent pulled events to the DDS infrastructure and are responsible for pushing the events to the subscribers. Since the DDS storage is assumed to be highly scalable and can achieve high throughput rates, all necessary modes for a pub/sub-based system are given. This approach might be useful in a distributed server environment. Also resilience is supported, which is not considered in this work. The evaluation of the approach can easily be done by applying our proposed methodology.

A general overview on requirements in filtering and its corresponding data models is given in [42] by Mühl. A generic “content-based data model”, as well as constraints for values and notification types are introduced. Also it defines a differentiation of perfect and imperfect merging of subscriptions, by considering similarities within their interests. These merging mechanisms are proposed with regards to optimized filter matching strategies. A detailed analysis of the imperfect and perfect matching is presented in [72] and in the PhD thesis “management of uncertainties in publish/subscribe system” [106] by Liu. The work of Liu focuses on fuzzy matching of event information against the interests a client subscribed for, which is a prerequisite for enabling semantic publish/subscribe systems.

Designing efficient filtering processes is one of the key issues in designing content-based routing. However, our focus is not on designing such filters, but rather on evaluating the filtering performance in varying scenarios, and to develop a flexible modeling approach for system dimensioning of a pub/sub system.

Distributed Event Brokers

Besides filtering performance at each event broker, it is necessary to keep the system scalable to distribute the overall load upon multiple brokers. In case multiple brokers are involved, subscriptions have to be distributed throughout the network, in order to optimize message routing and load. In [78], an efficient way to aggregate subscriptions and to select the routing path on application-layer is introduced. Subscription aggregation is also useful in a single server scenario, since the evaluation of a filter takes a noticeable amount of time. As our results showed, different broker applications can benefit from such optimizations. An efficient event routing for content-based publish/subscribe systems is proposed in [65]. This work introduces an architecture called Kyra which tries to balance the filter matching and routing load upon multiple brokers. Therefore the authors propose different routing approaches and evaluate these algorithms by comparing different parameters, e.g., network performance, storage cost, and processing load. The evaluations are done by simulative studies. A survey on common "routing algorithms for content-based publish/subscribe systems" is presented in [98]. A general overview on the entities as well as a set of routing options for event dissemination is given. Also architectural options for organizing subscription propagation and partitioning the filter matching mechanisms are discussed. The authors conclude that many of the presented algorithms are only evaluated by their authors using only "synthetic data sets" in simulative studies. This is a motivation for our experimental driven approach to evaluate the system performance presented.

Another approach to increase system throughput performance can be achieved by reducing the traffic at the server, which is called *quenching*. This approach is implemented by *Elvin4* as described in [29] and [38], a general-purpose notification service. In order to reduce network traffic, the routing entity informs all publishers periodically about all registered subscriptions. In this work, a theoretical estimation is presented, regarding performance differences between filtering on subscriber side and publisher side. The authors of *Siena* [34] propose a distributed environment, where the replication of the messages is done as close as possible to the desired subscriber. Consistently filters introduced by the subscribers are aggregated as far as possible. Our evaluations showed that in a data center scenario the network capacity is not a bottleneck for messaging with typical payload sizes, which might be different in an Internet scale scenario.

Hsiao proposes a different approach besides application-layer routing in [59] by considering IP multicast. The approach is based on the JMS framework and provides the interfaces as required by the JMS API. The organization of message-flows is mainly done by topic-based IP multicast groups. Complex filtering is done on subscriber side. Also a rate control is implemented, in case a subscriber gets overloaded. An implemented prototype showed promising results regarding the throughput with larger message body sizes. Following the approach in [70], the server has only to evaluate the header parts of a JMS message, what is described as “lazy deserialization” in the work of Koao and Hung. During normal deserialization process the overall JMS message Java object has to be restored in memory, which is a time and memory consuming task. Since the message body is not of any use for the JMS API this step can be omitted. We also evaluated the throughput capacity depending on the message body size, and found, that the observed impact of the message body size is well predictable, and does not depend on the server’s implementation, but on its network packet forward capacity.

Approaches for Generalized Models

In [63] an approach to model and validate distributed architectures based on the publish/subscribe pattern is presented. In modern and complex distributed systems, the automated model checking becomes an important part of developing and programming, since cascading effects might influence the system performance. In this work, we test JMS framework features in an isolated manner on a single server environment. The use of a single server environment reduces the difficulties in verifying the environment.

A state-persistent model for handling subscriptions is introduced in [47]. It describes events and subscriptions as points in a multidimensional space where the distance between points determines a match between an event and a subscription. This approach supports semantic-aware pub/sub systems. In addition to the architectural considerations and definitions, a pseudo implementation of the matching algorithm as well as an analytical evaluation of the expected performance is presented in [60]. A major focus of this work is on updating subscribers interests which means a modification of the subscription. The JMS framework does not support a subscription modification, so our work does not consider modification. We assume in our scenarios a consecutive deletion and a modified re-subscription process for processing a modification.

A cost model for publish/subscribe with focus on mobile grid environments is presented in [74]. The authors compare a client-server based and a polling based communication pattern with the publish/subscribe communication pattern by a cost-based analysis. The evaluation is done regarding the total cost for a given scenario and the cost for each single access to the system, in case of the pub/sub system is transmitting a message. While developing a pub/sub system representing its central nature in the software communication the calculation of costs is an important issue. Since its costs add to the ones introduced by the other software components it might take an important role in the overall service delivery. We focus not on evaluating costs, their models differ from our throughput analysis, but cost evaluation may complete the system analysis.

Peer-to-Peer Based Event Brokers

A major advantage for efficiently implementing the decoupling features in pub/sub environments is the central nature of the mediation platform, that all clients have to connect to. The well-known *peer-to-peer* (P2P) principle can add additional features to enable an efficient distribution of the pub/sub system. A lot of research has been done in this area. A general discussion of infrastructure-less pub/sub systems based on P2P principle is done in [82]. In [62] a publish/subscribe system based on a *distributed hash table* (DHT) is proposed. In the case of a DHT, all clients are involved in storing the subscriptions and evaluating them. Fault tolerance is introduced as a feature of the DHT approach. If a single subscription is frequently used, the associated node might get overloaded which limits the scalability of the overall system. Triantafillou et al. [77] also propose a DHT based system but specialize the setup to *Chord*, which forms the DHT in a ring-like overlay structure. The ring like overlay structure supports a better load balancing between the nodes, as their estimation of the workload shows. From a message throughput performance point of view, the overall achievable throughput in P2P based publish/subscribe architectures depends on the number of involved nodes and their CPU capacity. Typically, the available system resources in terms of CPU and memory are rather low in a fully distributed environment, therefore a large number of nodes is necessary. This leads to the former described problems in efficient subscription handling and message routing. All in all, the central approach, as focused in this work, is the better choice to cope with the requirements of a data center application. Some of the investigated algorithms might be useful in case of load balancing the servers.

Quality of Service Considerations

An important aspect in the field of publish/subscribe is the *quality of service* (QoS). Especially if a timely delivery of messages is necessary the publish/subscribe system has to be QoS aware. In [84] and [85] different approaches for enabling QoS guarantees are presented and evaluated in terms of their performance. Our focus is not on the QoS features of a publish/subscribe system, but we present an approach to model the internal behavior of a JMS server, which might be used to evaluate the desired QoS behavior.

Reliability Aspects

Introducing reliability in distributed and dynamic publish/subscribe networks causes additional overhead. In [55] and [67], different approaches to design a reliable publish/subscribe system are introduced and evaluated by simulations regarding different failure rates, e.g., network link failures. This applies for dynamic broker networks distributed over an Internet-based publish/subscribe network. In our data center-oriented approach, this kind of problem can be handled by adding redundant server resources with the appropriate resilience mechanisms. The resilience mechanisms in a data center might consider the presented approaches in the dynamic environment, but typically they introduce too much overhead compared to simple one-by-one server backup strategy.

Security Issues

Security and confidentiality are important issues for performance of a publish/subscribe system. In [51], the authors point out some security issues arising from the structure of pub/sub systems. A number of problems come along with this type of communication, e.g., authentication of publishers and subscribers among each other, information confidentiality, and subscription confidentiality. It must be taken into account whether the publish/subscribe infrastructure is trusted or not, resulting in different mechanisms that can be used to solve the mentioned problems, as described by the authors of [69]. Also, some of the security requirements conflict with the pub/sub model. For example, content-based routing includes the evaluation of the information. This is not possible if the message is encrypted (information confidentiality), except a trust relationship exists for the overall environment, including the mediation platform. Some of the approaches use a *public key infrastructure* (PKI) or similar approaches which are not part of the publish/subscribe domain, others rely on a trusted infrastructure. Since we are focusing on a data center scenario, our tests do not consider any security options, but can be easily extended to take them into account.

Applying Pub/Sub to Lower ISO/OSI Layers

In Future Internet scenarios currently under discussion, the publish/subscribe communication pattern is also of interest in replacing the current communication based on IP addressing. A project called *Publish-Subscribe Internetworking Routing Paradigm* (PSIRP) [103] focuses on this topic. Several scientific publications introduce different aspects of the routing paradigm on lower layers. In [103], a general overview on the PSIRP architecture is given, whereas [110] focuses on an experimental driven approach to evaluate a PSIRP prototype in current Future Internet test facilities, like PlanetLab [61] and FEDERICA [93]. Another approach with the same goal is [96] the LIPSIN environment, where a linespeed supporting publish/subscribe system is introduced. LIPSIN can be considered as another underlying forwarding fabric to IP, similar to Ethernet. The evaluations of the implemented prototypes showed similar delay performance just like a standard Linux-based IP-router. Since our application scenario considers the communication of software components the shift of the publish/subscribe communication to lower layers is an optional step which might be useful in future network environments.

Summary

The previous section discussed related work introducing new approaches to implement an efficient publish/subscribe enabled service. But in typical environments, a system is dimensioned and installed, based on evaluations of existing pieces of software and hardware. To get an idea how the system performs under varying load conditions and in different use-cases, a performance evaluation of a system is necessary which is the focus of the next section in related work.

2.4.2 Benchmarking Approaches for Message-Oriented Middleware

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.

JMS Benchmarks

The throughput performance of four different JMS servers is compared and evaluated in [71]: FioranoMQ [117], SonicMQ [132], TibcoEMS [143], and WebSphereMQ [122]. The study focuses on several message modes, e.g., the durable, persistent message transmission mode. But it does not consider filtering, which is the main objective in this work. The authors of [56] conduct a benchmark comparison for the Sun OneMQ [139] and IBM WebSphereMQ [122]. Additionally, there exist several other performance studies, with the focus on providing comparisons of the server capacities, like a comparison of

FioranoMQ and the SonicMQ [118], or the SonicMQ vs. the IBM WebSphereMQ series [134]. Also some of the vendors provide their own suites for testing the JMS server performance, like done by IBM Hursley [123], Sonic test harness [133], or JBoss [125]. The results of the different test suites consider the throughput performance in various message modes and, in particular, with different acknowledgment 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 performance models to forecast the maximum message throughput for given application scenarios.

The Apache working group provides the generic test tool JMeter for throughput tests of the ActiveMQ [113]. However, it has only limited functionality such that we rely on an implementation developed by us to automate our experiments.

Another benchmarking testbed is implemented by Pang [50]. It is designed to compare two message-oriented middleware servers, namely TIB/RV and SonicMQ. The tests evaluated the system's capacity considering message throughput, memory consumption, and CPU utilization. Also some stability tests are performed, to test the system under high loads and resource utilization. The tests consider the publish/subscribe pattern, as well as the point-to-point communication. In comparison to our tests, the introduced load is an order of magnitude lower which prevents the authors from detecting some effects like server crashes with unlimited message publishing. Also our tests focus on the filtering performance of the publish/subscribe engines since that is the crucial part of the system.

In [102], the authors introduce an approach to benchmark JMS in general. For their evaluations, they designed a message load considering the traffic mix of a standard supply chain as observed in a supermarket scenario, which is described in [91] and [90]. An extension of the JMSspec2007 benchmark by additional load scenarios and evaluating a current version of the ActiveMQ server is presented in [101]. The process of the benchmark considers similar aspects as provided in our experimental evaluations, like message size and number of consumers. The considered server software is in line with the systems we evaluated and leads to comparable results. In contrast to their work, we do not provide a standardized scenario with a fixed set of parameters. Fixed parameters are required by the JMSspec2007 benchmark for reasons of comparability with not yet evaluated servers. Our approach supports a flexible detection of bottlenecks in adapted scenarios. Additionally, we provide models to forecast the throughput performance of the system for a given scenario and investigate the impact of the internal queuing behavior of the server software.

Generalized Pub/Sub Benchmark Approaches

Another proposal for designing a “Benchmark Suite for Distributed Publish/Subscribe Systems” is presented in [45] but without measurement results. The setup of our experiments is in line with these recommendations. General benchmark guidelines were suggested in [52] which apply both to JMS systems and databases. However, scalability issues are not considered, which is the intention of our work. Baldoni et al. present a generalized mathematical model for a general pub/sub scenario in the durable mode with focus on message diffusion without filters in [54]. The authors enhanced their analytical models and validated it by simulations in [79]. In the work of Baldoni also the filter update intervals are considered, since their focus is on modeling a distributed event-based system. In our work, a mathematical model is presented for the throughput performance for a single server in the non-durable mode including different filter configurations. We evaluated our modeling approach by measurements and found it as valid and in line with the work of Baldoni.

In [80], the memory requirements of different filtering algorithms for pub/sub systems were studied theoretically and experimentally. Memory consumption during message processing is a key performance issue. In our experiments, we therefore carefully observed the memory usage, so it does not present a performance bottleneck in the system.

In the context of mobile networks, a publish/subscribe system benefits from the asynchronous communication. A performance evaluation of JMS servers regarding the throughput is presented in [68], where the authors emulate different network conditions typical to wireless networks. Since our network conditions remain constant, we do not see any effects like slow consumers, moving subscribers, or changing network conditions. However, the experiments conducted in this work can be extended to consider varying network conditions.

As already stated in the design oriented section of related work, QoS guarantees in a pub/sub network are a good approach to ensure performance. In [66], an empirical approach is considered to evaluate, if a JMS server is able to hold the chosen configuration and requirements set by the clients. In our work, we do not explicitly evaluate QoS parameters, but mention during the measurements, where the observed behavior differs from the one configured.

An experimental driven evaluation of two different JMS solutions, namely JBoss MQ and is done by Laranjeiro et al. in [94]. Their focus is on robustness, which is part of benchmarking evaluations, and security, where they evaluate the servers by different well-known problems in this field to show vulnerability and the protection against it. Since our experiments do not focus on security aspects, we switched off any features regarding security. We assume, that in a closed data center network the security aspect can be handled by other entities in the system. The robustness aspect is a crucial aspect and covered by the stress-testing approach we followed. By using our models we can predict the performance limits of the software, in order to prevent failures during design of an application.

2.4.3 Other Messaging Approaches and JMS Servers

For the evaluation part we selected a set of JMS servers as described in the following experimental chapter Chapter 3 of this work. Overall, we consider not only the selected servers. Several other servers and approaches were tested and discarded, since they crashed in our test scenarios, or did not provide the desired feature set. But still, they might be interesting approaches in the publish/subscribe application middleware field.

Messaging Product Vendors

That there are major differences in performance and focus between the different vendors becomes apparent if we look at the history of several software products. For example Oracle completed their portfolio be acquiring the companies Bea and Sun including their messaging servers. The former products of Bea and Sun are still available as standalone software in the product portfolio of Oracle. In the area of open source the development of the ActiveMQ server, which is part of our evaluations, moved from a small company to the Apache Foundation, where it matured to a industry accepted application. The former IBM research project Gryphon [31], which included a large set of mechanisms developed in science, became part of the IBM WebSphereMQ server. The WebSphereMQ version including all optimizations introduced by Gryphon is part of our performance evaluation.

Other major vendors, like Amazon, do not focus on the JMS framework, while implementing own versions of a large scale publish/subscribe system. The *Simple Notification Service* (SNS) with focus on real time event communication and the *Simple Queue Service* (SQS) focusing on the decoupling of communication by introducing a queue, are good examples for highly scalable publish/subscribe systems. Since these services are only available as a cloud service and not as a standalone product, we did not consider them in our research.

Application-Layer Communication

An important part while using pub/sub systems is the underlying communication protocol. From a vendor point of view a large variety of communication protocol standards have to be supported to reflect the different needs of the customers and guarantee their interoperability. But the selection of the application-layer communication protocol has a high impact on the overall performance of the system, e.g., if a message is transmitted in plain text, or serialized to a binary data stream. In our scenarios we used the application-layer transport protocols, as configured by default. We payed attention to consider only protocols, which transmit the messages in a binary stream and do not include text based protocols, like JSON

[105], SOAP, HTTP, or even e-mail. One of the most common protocols used for binary data transmission is the so-called *Advanced Message Queuing Protocol* (AMQP) [104]. AMQP describes an open protocol specification and defines efficient application message formats, which are called “wire-level” formats in the AMQP documentation. The AMQP documentation describes some semantics of broker services which is in competition with the JMS framework. Many vendors, like the ActiveMQ software, use the AMQP protocol specifications for application-layer transport and offer the interfaces defined by the JMS framework to the customer applications. The RabbitMQ [108] software as a recent product fully supports the AMQP framework while being based on the JMS framework. Since RabbitMQ does not directly offer JMS adapters, we prefer the ActiveMQ software as an open source representative in our evaluations. However, our tests can easily be extended to test the RabbitMQ software as well. To increase the performance by using alternative physical connection technologies an approach is introduced in [95] where the AMQP protocol is applied on InfiniBand, a very powerful transmission technology using copper cables.

Other JMS Servers

There is a huge amount of industry standard JMS brokers available on market. A list of current implementations can be found at [109]. For our evaluations, we consider the server implementions of FioranoMQ [116], IBM WebSphereMQ [122], Bea WebLogic Application Server (now Oracle) [107], Sun Java System Message Queue (now Oracle) [139], and ActiveMQ [112]. We also look at different JMS middleware software, like JORAM [99], JBossMQ [100], and open source implementation OpenJMS [92]. JBossMQ (version 4.0.3SP1), in the version we have tested, crashed while handling the default load of our test clients, as described in the next chapter Chapter 3. Therefore we implemented a special rate control in our test clients, to perform our experiments, but since they do not fully apply to our test requirements we do not consider them in this work. OpenJMS completes our test scenarios successfully, but its overall performance is too low to be considered in our results.

General Pub/Sub Prototypes with Focus on Research

In contrast to the previously introduced industry focused software components, there are multiple research projects providing prototypes and grown up environments to test and support new approaches in science.

One of the first implementations of a content-based pub/sub system is the *Scalable Internet Event Notification Architecture* (SIENA) [30] and [137]. The focus of the SIENA system is on an Internet-scale deployment, with efficient overlay connections for routing events and subscriptions. SIENA is a very flexible and extensible approach, but not of use in a productive environment. The detailed description of the event handling introduced by SIENA supports us by designing efficient filtering tests. Since SIENA does not support JMS we do not consider it in our evaluations.

Another flexible approach for a distributed event-based middleware is introduced by Pietzuch in [75] and called HERMES. A characteristic of the presented work is the integration of an evaluation of the introduced algorithms by a distributed simulator. We do not simulate our scenarios, but present analytical models derived from our experiments.

The *Java Event-Based Distributed Infrastructure* (JEDI) [40] organizes the distributed brokers, called *event dispatchers*, in a tree like structure. Each subscription is propagated upwards in this tree structure to the root element. This kind of architecture supports a dynamic organization of event dissemination, but lacks robustness. We did not consider this approach since it implements only a very simple not-optimized filtering engine. Another reason not to consider the three approaches, SIENA, HERMES, or JEDI is that they do not support the JMS framework.

SpiderCast, introduced in [89] by Chockler et al., is a topic-based distributed pub/sub system focusing the optimization of well-correlated subscription patterns using a simple distributed heuristic. In [89] the results are mostly evaluated on an empirical level while the authors provide a theoretical approximation of their approach in [88]. Overall, this approach introduces a very efficient distributed pub/sub system, but is limited to the topic-based pub/sub communication pattern.

One of the few scientific prototypes including adapters for the JMS framework is the Narada Brokering service, introduced in [46]. Since the focus of this approach is on a distributed broker service for grid computing and Narada implements an XML-based text-oriented message processing, we do not consider it for our server evaluations. Pereira et al. introduce another XML-based approach in [43]. The authors claim that this approach is high performant in terms of message throughput. However, the performance evaluations done in the work of Fox and Pereira show that the achievable message throughput performance is in the medium range of the systems we evaluate.

Corona, which is presented in [86], is another distributed pub/sub system with focus on polling based message reception by the subscribers. An interesting part of the work is the evaluation of the system in the PlanetLab testbed, which is distributed all over the Internet. However, we consider a local testbed to ensure repeatability of our experiments regarding the message throughput performance. This cannot be guaranteed by using the PlanetLab environment while it supports a large-scale test of software functionality.

Summary

All approaches and methods in this section help us to understand the key performance issues in a publish/subscribe system. Based on the literature, we start out to design a testbed for evaluating critical parameters of a centralized publish/subscribe system based on the JMS framework. The selection of the evaluated servers is based on the products available on the market, supporting our data center-oriented scenario, and representing a wide range of different implementation options, which leads to different performance characteristics during evaluation. In the following we consider the server implementions of FioranoMQ [116], IBM WebSphereMQ [122], Bea WebLogic Application Server (now Oracle) [107], Sun Java System Message Queue (now Oracle) [139], and ActiveMQ [112]. The next chapter introduces the experimental environment, which enables the message throughput measurements, and is the basis for our modeling and analytical approaches.

3 Experimental-Based System Evaluation

In this section, we use the maximum throughput based on measurements of different JMS servers as a performance measure. The objective is to assess and characterize the impact of specific application scenarios on the server performance. In particular, we consider different filter scenarios, as they are essential for the use of a JMS server as a general message routing platform. We explain the experimental facility setup and conduct several parameter studies to explore their impact on the JMS server throughput.

3.1 Experimental Environment and Experiment Design

The objective of this section is the assessment of the message throughput of different JMS servers, supporting the publish/subscribe communication pattern, by measuring the performance under various conditions. Since the implementation of the JMS framework is up to the vendor, we build a testbed and design a set of experiments to evaluate the server performance characteristics. This approach identifies system critical parameters which might present bottlenecks.

For comparability and reproducibility reasons, we first describe the hardware components involved, the network setup, and the configuration of the operating system. Then, an overview of the considered JMS server implementations and their configuration is given. Finally, we describe the critical parameters, the overall experiment design space, and the measurement methodology.

3.1.1 Experimental Environment Setup

For our experiments, we use dedicated hardware and a proprietary test client software. The experimental environment is defined by a workflow, which is partly automated.

Hardware Setup

Our dedicated test environment consists of a number of 12 computers as illustrated in Figure 3.1. Up to 10 of them are client machines and two are used for control purposes, e.g., controlling jobs like setting up test scenarios and starting measurement runs. The 10 client machines have a 1 Gbit/s network interface which is connected to a single Gigabit switch. They are equipped with 3.2 GHz single core CPUs and 2 GB system memory. The operating system is openSuSe Linux 9.1 in standard configuration. To run the JMS environment, we install the Java SDK 1.5.0 in default configuration. The control machines are connected over a 100 Mbit/s interface to the Gigabit switch. The hardware and installed operating system is tested by performing a system level benchmark which is repeated in case any hardware component or system software changes. The results of the benchmark are stored as default values and the state of the system is called *calibrated*.

Design of the Test Client

In our experiments, one *server machine* is used as a dedicated JMS server. Up to four *publisher machines* are exclusively used to run publishers and one or up to 8 *subscriber machines* run the subscriber applications depending on the experiment. If two or more publisher or subscriber machines are used, the emulated publishers or subscribers are distributed equally between them. We implemented test client software in such a way that each publisher or subscriber is realized as a single Java thread, which has an 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.

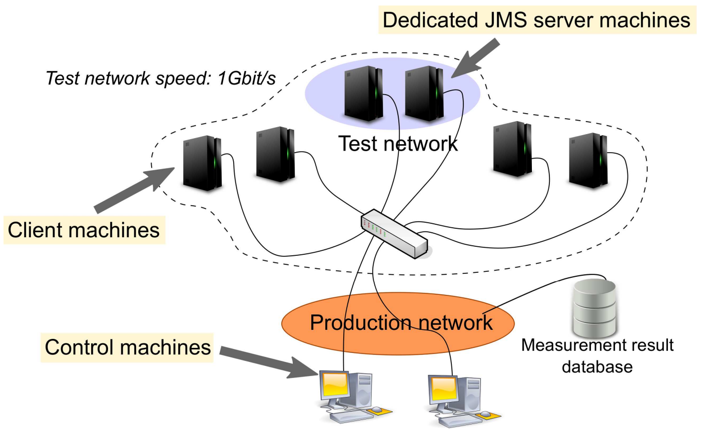

Dedicated JMS server machines
Test network speed: 1Gbit/s
Test network
Client machines
Production network
Control machines
Measurement result database

Figure 3.1: *Hardware and network setup in the testbed.*

Workflow of an Experiment

In general, an *experiment* is a set of single *measurements*, where only one parameter is changed, while all others are left unchanged. Each measurement is repeated several times as *measurement runs* to increase confidence in the resulting mean value. The different measurement runs are used to calculate the standard deviation and the corresponding confidence level. Figure 3.2 depicts a chronological overview of three different phases from the experimental setup over conducting the measurement runs up to the analysis of the resulting data. Phase 1 considers the design of a single or complex experiment, and defines the parameters to evaluate. In case a experiment requires redesign, a manual interaction is necessary. This can occur if a certain configuration and the corresponding measurement runs always lead to a failure of the overall experiment, which might be the case if the JMS server faces a software error.

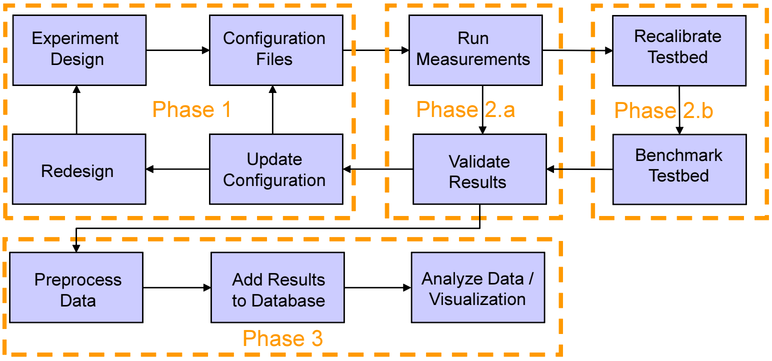


Figure 3.2: *Measuring and experimenting workflow.*

Automation of the Experiment Workflow

The next step is to create and deploy configuration files for the automated test environment. Phase 2 consists of two subordinated tasks. Each measurement produces results, which are automatically validated by the test environment considering the constraints defined for the experiment. If a violation of a given constraint is detected, the results of this particular measurement run are rejected, and a re-run with an updated configuration is scheduled. On a regular basis or if the constraints are violated multiple times, the testbed re-calibrates itself, by rebooting all machines and conducting a system level benchmark of each machine. The result of the benchmark is again validated against a previously measured default value. In case the benchmark result validates negatively, a dump of all system relevant data is performed, e.g., log-files and I/O counters. The measurement is set on halt for manual interaction. Additionally, a new default benchmark value is generated after each system upgrade, e.g., after introducing new security patches. During the last phase, a post-processing of all data is performed, in order to make them available via a database interface or as result figures, e.g., as shown in this work.

3.1.2 Evaluated JMS Server Environment

For our performance evaluations, we focus on existing software implemented by different vendors. All these vendors have slightly different implementations regarding the pub/sub communication pattern. Therefore, the different software products are shortly introduced in this section. The installation and configuration specialities of the five considered server types are briefly described, where the basic idea is to keep the default configuration when possible.

Table 3.1: *JMS vendor: Overview on evaluated JMS servers.*

Vendor	Product name	Version	Open source
Fiorano Inc.	FioranoMQ	7.5	No
Sun Microsystems	Sun Java System MQ	3.6	No
IBM	WebSphereMQ	6.0	No
Bea Inc.	WebLogic App. Server	9.0	No
Apache Foundation	ActiveMQ	4.0	Yes

Fiorano MQ

The *FioranoMQ* [117] version 7.5 server components is installed as JMS server software. The vendor's default configuration is used as delivered with the trial version. The server has to be executed with superuser permissions; otherwise user restrictions can limit the number of simultaneously connected clients to the FioranoMQ kernel.

Sun Java System Message Queue (SunMQ)

We install the *Sun Java System Message Queue* 3 2005Q1 platform edition (version 3.6) [139], which is shipped with a trial license including all features of the enterprise edition. We use its default configuration except for the following modifications. To enable the publish/subscribe mode, we set up a customized default topic. Normally, a large buffer is reserved for incoming messages.

However, as it is too large for our experiments, we limit it to a maximum of 10,000 messages and switch on the flow control to avoid message loss at the incoming buffer. Otherwise, if the incoming message buffer size is set to the default value, the server starts dispatching messages to the subscribers after receiving all messages from the publisher since we send them in a saturated manner. Additionally, we increase the maximum threshold for simultaneously connected subscribers from 100 to 400.

IBM WebSphere MQ (WebSphereMQ)

We install the *IBM WebSphere MQ 6.0* trial version [122] on the server machine with the default configuration except for the following modifications. For performance reasons, we disable the security module, since our experiments do not focus on security issues. We raise the internal restriction regarding the number of parallel connections to the queue manager from the default value 100 to 500. The WebSphereMQ software offers to use a third party pub/sub engine. To conduct our experiments, we use the WebSphereMQ's integrated pub/sub feature.

BEA WebLogic Application Server

For evaluation of the *BEA WebLogic Application Server* (version 9.0) [107] we use the evaluation version provided by BEA as a binary. We adjust the heap-size of the Java Virtual Machine, such that 1 GB of memory is available for the server. Furthermore, we define the upper and lower bounds for the size of the internal message queue, which triggers the internal flow control. This becomes necessary, because the server does not slow down the publishers in default configuration, even with an exhausted queue space. This leads to unpredictable behavior and the server software fails.

Apache ActiveMQ

Finally, the open source product *ActiveMQ* is considered for evaluation. The ActiveMQ software is maintained by the Apache software foundation. Besides the official documentation [112], an insightful description for ActiveMQ is available as work-in-progress in [136]. In our tests, we evaluate and compare several versions of the ActiveMQ software, also in varying configurations. This helps us to separate software bugs from software performance behavior. For the results presented in this work, we focus on the versions 4.0 and 4.1. ActiveMQ relies on system files to maintain connections and software availability states. To run ActiveMQ with larger numbers of publishers and subscribers, it needs more system file handles at the same time than the Linux kernel allows by default. We increase the number of allowed parallel file handles in the operating system to an arbitrary chosen value of $16{,}384$ compared to the default value of $4{,}096$. This value proved to be sufficient for all experiments.

3.1.3 Measurement Methodology

Our objective is the measurement of the JMS server message throughput capacity. Therefore, we load the JMS server in all our experiments close to 100% CPU load and verify that no other bottlenecks like system memory or network capacity exist on the server machine. The publisher and subscriber machines must not be bottlenecks, i.e., they should not run at an average CPU load exceeding 75%. This ensures that there is enough switching capacity left for the emulated publishers or subscribers. To monitor these side conditions, we use the Linux tool "sar", which is part of the "sysstat" package [119]. We monitor the CPU utilization, I/O, memory, and network utilization for each measurement run. We use a per second interval for recording our measurement values. Without a running JMS 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 96%, by holding a 95% confidence level. This means that the measured average of the JMS server's CPU utilization during a measurement run should have a mean value which does not violate the 95% confidence level. To illustrate a setting of a typical experiment, Figure 3.3 shows the CPU utilization of the publisher, subscriber, and server machines. We observe that the CPU utilization for the JMS server machine remains above 95% at any time during the measurement run which is the desired behavior.

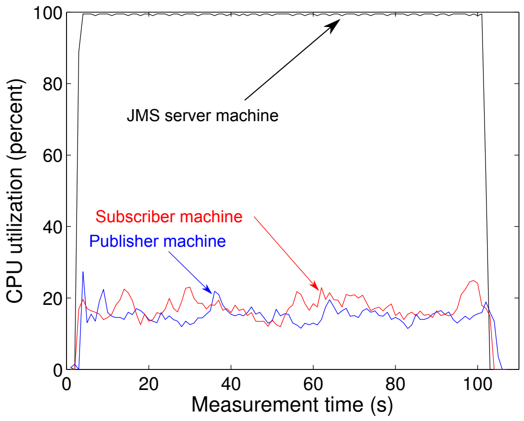


Figure 3.3: *CPU utilization of a typical measurement run.*

Experiments are conducted as follows. The publishers run in 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 since publisher side message queuing is used. To save system processing resources during the measurement phase, all JMS messages are created in advance. For the same reason, all necessary network connections are established before the measurements are taken. As depicted in Figure 3.4, the control machine initiates the system monitoring at first. After that, the server components are initialized, followed by the subscriber, as passive receiving elements. If they are successfully up and running, the publishers are initialized to standby. After a short period and verification of system readiness, an execute command is sent almost simultaneously to the publishers and they start sending their predefined messages.

Each experiment typically takes 600 s. Several experiments showed that a shorter experiment duration might include only warmup effects. Since we observed warm-up and cool-down effects, we cut off the first and last 50 to 100 s,

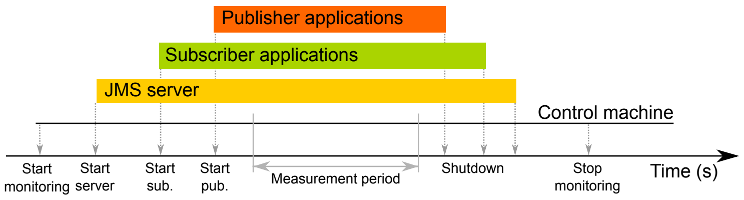


Figure 3.4: *Schedule of a measurement run.*

depending on the results of the system calibration. We count the overall number of sent messages at the publishers and the overall number of received messages by the subscribers within the remaining 400 to 500 s interval to calculate the server's rate of received and dispatched messages. For verification purposes, we repeat the measurements several times. In most cases, if we received overall valid measurement data within our constraints, their results hardly differ such that confidence intervals are very narrow even for a few runs.

Messages arriving at the JMS server sent by the publishers are called *received messages* and the ones sent out to the subscribers are called *dispatched messages*, respectively. The sum of received and dispatched messages is denoted by *overall message throughput*. If a message sent by one publisher is dispatched to r different subscribers, it is replicated and sent r times by the JMS server and we call r the *replication grade* of the message.

3.1.4 Experiment Parameter Design Space

In Figure 3.5 we depict a categorization of the different parameters, which impact the JMS server performance. This applies also for designing and dimensioning a JMS server deployment in practice. Each of the four main topics *server utilization*, *information granularity*, *network/subscriber utilization*, and *throughput* represent a domain of parameters, which are either adjustable by the designer or given by the desired application scenario. Our goal is, by using a common experiment design, to test each parameter in an isolated environment and vary it within in reasonable limits.

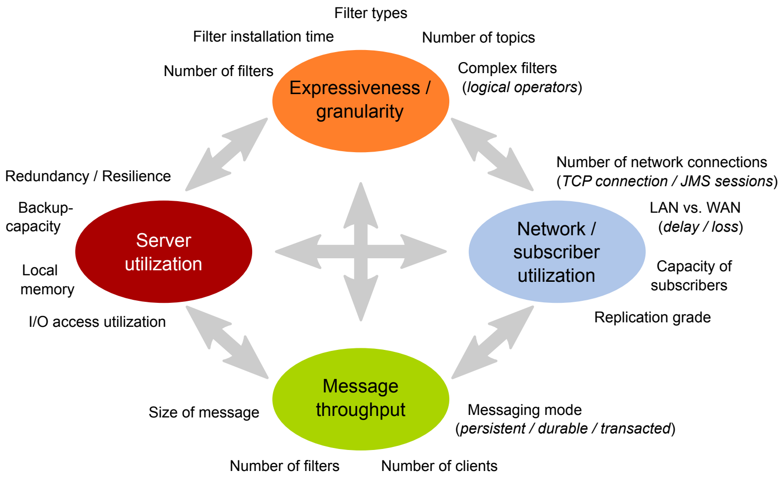


Figure 3.5: *Parameters and tradeoffs for JMS performance.*

A list of experiments is shown in Table 3.2. We start with some basic measurements in order to retrieve valuable input for our measurement setup, like the impact of the number of publishers and subscribers. Since our goal is to measure the soft capacity of the JMS servers, we had to determine the minimum number of clients necessary to fully load the server machine.

The chosen performance measure is the overall message throughput of the JMS server in terms of number of messages. Also the impact of a varying message body size influences the data throughput performance. Of basic interest is the performance impact, by enabling the filtering engine. This is done by an experiment considering simple filters. In realistic environments, the basic experiments help to identify the overall limits of the system. The performance of such a system depends mostly on the amount of information a subscriber likes to receive. This influences the complexity of the filter introduced to the system.

Table 3.2: *Overview on conducted experiments.*

Experiment	FioranoMQ	SunMQ	WebSphereMQ	Bea WebLogic	ActiveMQ
Basic Experiments					
Impact of publishers	x	x	x	x	x
Impact of subscribers	x	x	x	x	x
Impact of message size	x	x	x	x	x
Impact of topics	x	x	x		
Impact of simple filters	x	x	x	x	x
Complex Filtering					
Impact of AND-filters	x	x	x	x	x
Impact of OR-filters	x	x	x	x	x
Impact of IN-filters				x	x
Subscription / Connection Handling					
Impact of TCP connections				x	x
Impact of flash-crowds				x	x

Within the complex filtering experiments, we evaluate the joint impact of the number of involved subscribers and the replication grade controlled by a varying filter complexity.

In most systems, the subscribers are the dynamic elements, and the installation and removal of filters might influence the overall system performance. A high rate of system configuration changes, in terms of number of filters and connected subscribers, is also known as busy-hour or flash-crowd scenario. A flash-crowd scenario occurs typically after a system failure.

Another parameter is the observation that long running systems have an aging behavior. We performed several long running experiments, but the impact of the system aging is too specific for the used environment to draw some general conclusions. Thus, we present only results for a reasonable long experiment run time as described in the experiment setup. Reasonable means that we have to capture all short term effects, but slow long term performance impacts are not captured.

Since our major goal is not to compare the different vendors in a benchmarking manner, but provide a methodology to identify bottlenecks, we did not repeat each experiment for all considered servers.

3.2 Testing Basic System Performance

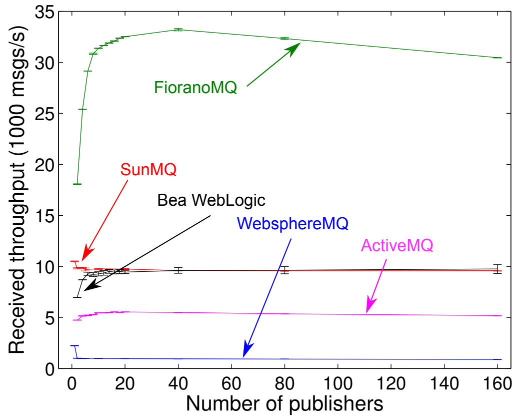
Using the experimental setup and parameters described in the previous section, we present in this section the results of the different experiments. The objective is to assess and characterize the impact of basic application scenarios in order to calibrate our experiments and to obtain knowledge about the limits of the examined JMS servers.

3.2.1 Impact of the Number of Publishers

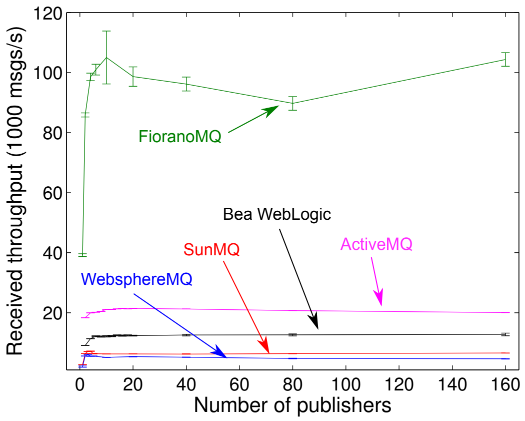
In our first experiment, we study the impact of the number of publishers on the message throughput. Two machines carry a varying number of publishers and one machine hosts a single subscriber. Figure 3.6(a) shows the received message throughput at the JMS server in the persistent mode, i.e., lost messages are retransmitted by the JMS server and messages are preliminarily written on a disk for recovery purposes. FioranoMQ achieves the highest received message throughput with 32,000 msgs/s, followed by SunMQ and Bea WebLogic with 9,500 msgs/s. ActiveMQ achieves about 5,500 msgs/s received message throughput and WebSphereMQ only 1,000 msgs/s. Thus, the message throughput spans several orders of magnitude. From the results we conclude that FioranoMQ requires 40 publishers to achieve its maximum throughput, whereas SunMQ and WebSphereMQ need only 5 publishers to achieve a typical throughput. For ActiveMQ and Bea WebLogic 10 publisher threads are sufficient to generate a typical message throughput. We assume a measured throughput as typical if its value is close to the maximum measured throughput and it covers a sufficiently large interval of the examined parameter. As a consequence, we consider in the following experiments at least 10 or more publishers.

To assess the impact of the non-persistent mode, we repeat the experiment runs with non-persistent messages where the server does not have to maintain a central storage. The results are collected in Figure 3.6(b). The received throughput is about $100{,}000$ msgs/s for FioranoMQ, $13{,}500$ msgs/s for SunMQ, and $9{,}500$ msgs/s for WebSphereMQ. ActiveMQ with $21{,}000$ msgs/s and Bea WebLogic with $6{,}000$ msgs/s also show an increased throughput. ActiveMQ can increase its throughput about 3.8 times, whereas Bea WebLogic can only achieve a factor of about 1.3. Thus, the message throughput is significantly increased, in particular for WebSphereMQ. However, especially for WebSphereMQ, we observe a high packet loss rate of about 8% under full load. All other servers discard less than 5% of the offered message load.

We repeat both experiment series at least five times and calculate the 98% confidence intervals on this basis. They are shown for all servers in Figure 3.6. For the persistent mode, they are very narrow for all servers which results from hardly varying system conditions. For the non-persistent mode, the confidence intervals are narrow except the ones for the FioranoMQ server, which might be a result of the outstanding increase of throughput between the persistent and the non-persistent mode. For our scenarios we cannot accept message loss, thus we consider only the persistent mode. We omit the presentation of confidence levels in the following figures for the sake of clarity. Nevertheless, our internal validation algorithm checked them after each experiment which guarantees that our constraints are not violated.



(a) Message transmission in persistent mode.



(b) Message transmission in non-persistent mode.

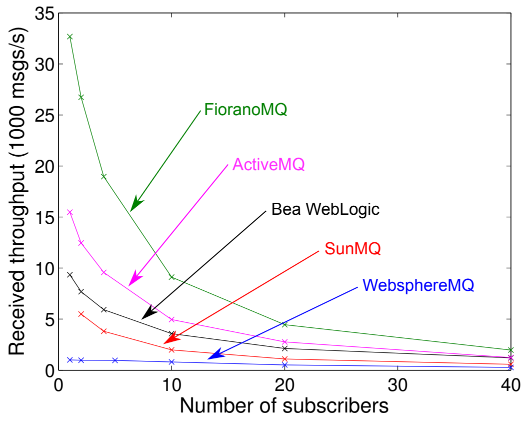
Figure 3.6: *Impact of the number of publishers on the received message throughput.*

3.2.2 Impact of the Number of Subscribers

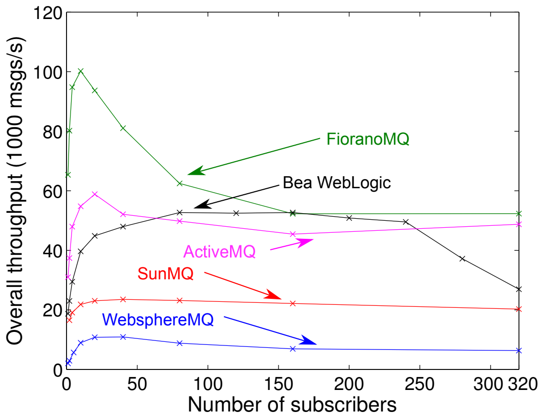
Similar to the previous experiment, we investigate the impact of the number of subscribers. To that end, we have 10 publisher threads running on one machine and vary the number of subscribers on two other machines. Figure 3.7 shows the received and the overall message throughput for different servers.

The received message rate, as depicted in Figure 3.7(a), decreases significantly with increasing number of subscribers m. Considering for example the Bea WebLogic this starts with a measured received throughput of $9{,}500$ msgs/s for one subscriber and ends with about 50 msgs/s for 320 subscribers which is omitted in Figure 3.6(a) for the sake of clarity. All other servers observe a similar decrease for the received message throughput. This can be explained as follows. No filters are applied and all messages are delivered to all subscribers. Thus, each message is replicated m times and we gain a replication grade of $r = m$ for this experiment. This requires more computational effort 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 remains constant. Hence, the replication grade must be considered when we compare the performance measures from different experiments.

Furthermore, it can be observed that the overall message throughput, as depicted in Figure 3.7(b), increases for all servers to an individual maximum of each server besides Bea WebLogic. If we consider the Bea WebLogic server, the overall message throughput starts with a value of $19{,}500$ msgs/s and increases it to a maximum throughput of about $53{,}000$ msgs/s for 80 to 160 subscribers. If the number of connected subscribers is larger than 160, the overall throughput decreases to a value of about $23{,}000$ msgs/s for 320 subscribers. According to our monitoring tools, the server CPU is still the only bottleneck in the system. This phenomenon can also be observed for all other server types but only to a minor degree. The reason might be that the persistent store of the messages introduces additional delays beside the pure I/O accesses, which is also discussed for ActiveMQ in the corresponding development community.



(a) Received message throughput.



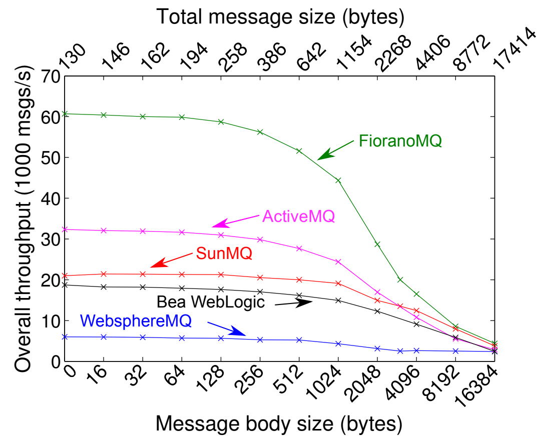
(b) Overall message throughput.

Figure 3.7: *Impact of the number of subscribers on the message throughput.*

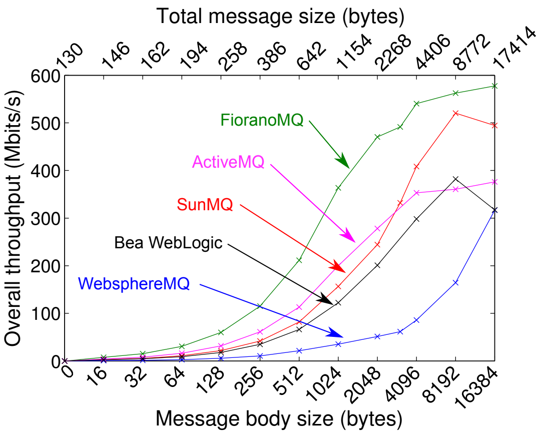
3.2.3 Impact of the Message Size

The throughput of a JMS server can in addition to the measure in messages per second (*message throughput*) be measured in transmitted data volume per second (*data throughput*). The message body size has certainly an impact on both values. A larger message payload increases the data throughput, but larger messages may also take more time for processing and reduce the message throughput. To quantify this tradeoff, we test the maximum throughput depending on the message size. For each server type, we use such a set up that the server achieves a sufficiently high throughput, i.e., 10 publisher threads on two machines. We use 10 subscriber on two machines for all servers. Figure 3.8 shows the overall throughput depending on the payload and the corresponding message body size. The calculation of the corresponding total 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. This value slightly varies for the different servers, since different application-layer transport protocols are used.

Figure 3.8(a) shows that an increasing message body size decreases the message throughput and increases the data throughput significantly, as depicted in Figure 3.8(b). For small message sizes with a body size of 0 bytes, the message throughput is limited by 61,000 msgs/s for FioranoMQ and 6,000 msgs/s for WebSphereMQ. For large message bodies of 16,384 bytes, the throughput is limited by 4,400 msgs/s and 2,400 msgs/s. Thus, the capacity ratio between the server types changes. The performance degradation of the servers has different shapes and depends also on the application scenario of the server, i.e., the number of publishers and subscribers, the message replication grade, and the filters. The overall consumed bandwidth is between 336 Mbit/s and 614 Mbit/s for the different servers. This is very large, but it does not yet reach the TCP transmission limit of our network for which we measured 820 Mbit/s in both directions. In our experiments, the default value for the message body size is set to 0 bytes. This reduces the influence of the systems I/O performance.



(a) Impact on the message throughput.



(b) Impact on the data throughput.

Figure 3.8: *Impact of the message body size on the overall throughput.*

We observe for basic network measurements without any application interference that smaller payloads decrease the maximum achievable network throughput. But even with small packet sizes the network does not present a bottleneck in our experiments. Hence, the effects of the filtering experiments are more significant.

3.2.4 Impact of the Number 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 two different replication grades. In our experiment, 10 publisher threads are installed on one publisher machine and two machines host up to 20 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. A subscriber can be registered to multiple topics at the same time.

Figure 3.9 shows the message throughput for FioranoMQ, SunMQ, and WebSphereMQ. Since the benefit of the message throughput results is limited for the topic scenario, we do not repeat the measurements for ActiveMQ and Bea WebLogic. In our experiments FioranoMQ achieves the highest throughput followed by SunMQ and WebSphereMQ. The throughput converges asymptotically to a value that is specific to the message replication grade. This value increases mostly with the replication grade. This holds true for all three server types. The throughput limit for many topics and a replication grade $r = 20$ amounts to $28{,}000$ msgs/s for FioranoMQ, $16{,}000$ msgs/s for SunMQ, and $4{,}000$ msgs/s for WebSphereMQ. Installing more topics on the servers leads to a decrease of the overall message throughput performance, which can be observed for all three servers. Hence, topics can be used for coarse message selection with a moderate performance loss for many topics. In particular, the impact on message throughput is weaker for an increasing number of topics than the message replication grade.

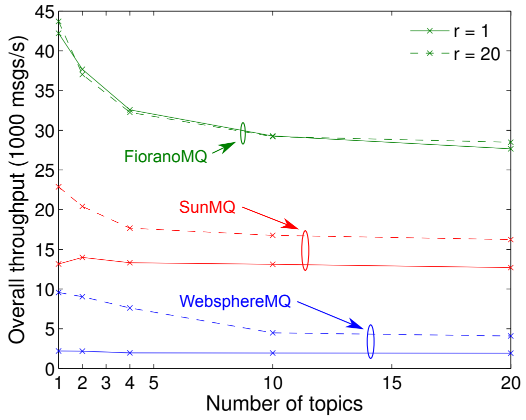


Figure 3.9: *Impact of the number of topics on the message throughput for different replication grades.*

3.2.5 Impact of Filter Activation

In the next experiment, we evaluate the impact of filter activation on the message throughput. Figure 3.10 shows the overall message throughput depending on the number of subscribers with and without filters. We used 10 publishers in all experiments.

FioranoMQ achieves its maximum throughput for 10 subscribers, about 100,000 msgs/s for all subscribers without filters, but only 36,000 msgs/s with application property filters, which are part of the dynamic JMS message header. We omitted the maximum throughput of FioranoMQ without filters in Figure 3.10 for the sake of clarity. ActiveMQ, SunMQ, and WebSphereMQ require both 20 to 40 subscribers to reach their maximum throughput of 59,000 msgs/s, 23,000 msgs/s, and 11,000 msgs/s, respectively. Bea WebLogic reaches its maximum throughput of 51,000 msgs/s between 50 and 200 subscribers. In contrast to FioranoMQ, all four competitors show only a slightly decreased capacity with

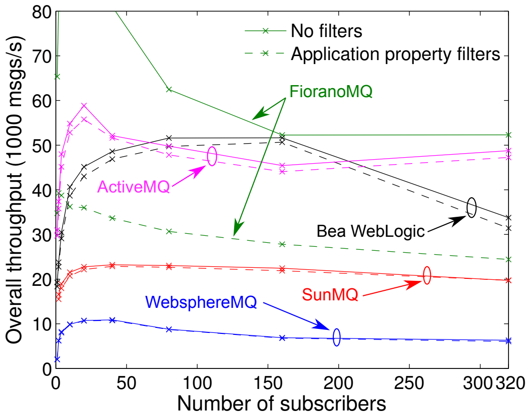


Figure 3.10: *Impact of filter activation and the number of subscribers on the message throughput.*

activated filtering. Thus, they are hardly slowed down by the filtering engine in this experiment. However, this finding is only valid if the message replication grade increases with the number of subscribers, which is a rather artificial case. In Chapter 4, we study the joint impact of filters and the replication grade for each server type in detail. After all, we learn from these results that at least 10 subscribers are required for future experiments to get a representative value for the maximum overall message throughput. As already argued for the number of publisher the choice of a typical message throughput also determines the number of subscribers for our further experiments.

3.3 Impact of Complex Filtering

A single client may be interested in a differentiated subset of messages which can be distinguished by a set of application header properties. Therefore, an enhanced filter, we call it *complex filter*, can be installed on the server. A complex filter consists of multiple simple filter components connected by logical operators, like ”OR” or ”AND”. The following section evaluates the impact of different complex filter types on JMS server throughput performance.

3.3.1 Impact of OR-Filters

If a single client is interested in messages with different application property values a logical ”OR”-operator is required. There are two different options to get these messages. The client sets up subscribers

(1) with a simple filter for each desired message type.

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

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

(1) To assess simple filters, we set up for each different ID exactly one subscriber with a filter for that ID.

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

In this experiment we use one publisher machine with 10 publisher threads and one subscriber machine with a varying number of subscribers or 5 subscribers, respectively.

Figure 3.11 shows the message throughput depending on $\frac{n}{5}$, which is the number of components in the complex OR-filter or the number of different

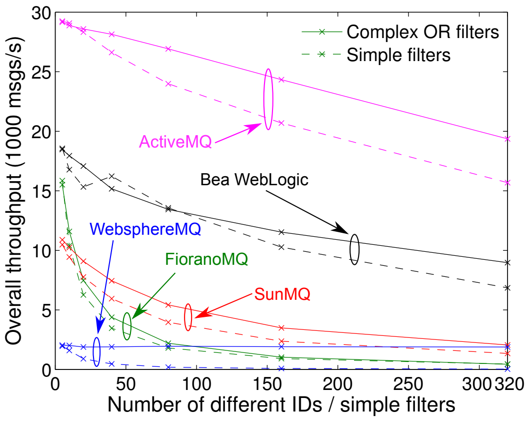


Figure 3.11: *Impact of simple filters and complex OR-filters on the message throughput for a replication grade of* $r = 1$.

simple subscribers per client. Firstly, we observe that the message throughput decreases significantly for an increasing number of installed simple filters. This is unlike in Figure 3.10 and the difference is caused by the smaller replication grade which is $r = 1$ instead of $r = n$. Thus, the number of filters decreases the message throughput considerably if the messages are not forwarded to all subscribers, which is usually intended to avoid with filters. Secondly, we observe that complex filters lead to a larger throughput than simple filters but the extent of the performance gain depends strongly on the server type. For FioranoMQ, complex filters lead to a slightly larger throughput than multiple simple filters per client.

For SunMQ, complex filters yield a performance gain of roughly 1,000 msgs/s. For WebSphereMQ, complex filters even avoid the performance loss that is observed for simple filters. Thus, the handling of simple and complex filters by WebSphereMQ takes the same computation effort. However, this finding holds doubtless only to a certain extent. ActiveMQ outperforms all other servers. The throughput for the simple filter experiment is mostly lower than the throughput for the complex OR filters experiment. Also for the Bea WebLogic the throughput achieved with complex OR filters is higher than the one with simple filters, except for 40 to 80 installed simple filters. This behavior can be explained by the results of the basic experiments, where the maximum throughput performance is reached at about 80 subscribers.

3.3.2 Impact of AND-Filters

In the application header part 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 filters but not for correlation ID filters. We use one machine with 10 publisher threads and one machine with $m = 10$ subscriber threads. The subscriber machines are numbered by $j \in [1; m]$.

We design two experiments with different potential for optimization of filter matching. The subscribers set up the following complex AND-filters of different length n, where P_x denotes an application property:

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

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

The corresponding messages are sent by the publishers in a round robin fashion to achieve a replication grade of $r = 1$. Then in experiment (1), the filters can already reject non-matching messages by looking at the first filter component. In experiment (2) the JMS server can only reject non-matching messages by looking at all n filter components. The experiments are designed such that both the replication grade and the number of subscribers remains 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 obtain the same number of filter components.

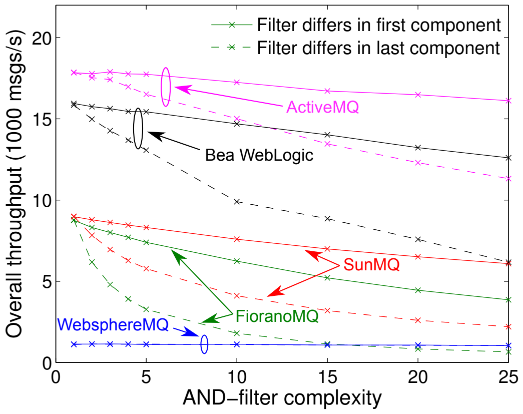


Figure 3.12: *Impact of an early non-match decision for AND-filters on the message throughput depending on the filter complexity.*

Figure 3.12 shows the message throughput depending on the filter complexity n. The filter complexity reduces the server capacity significantly for all servers beside the WebSphereMQ. Experiment (1) yields a considerably larger message throughput than experiment (2). Thus, an early reject decision of the filters shortens the processing time of a message and increases thereby the server capacity. As a consequence, programmers should care for the order of individual components within AND-filters: components with the least match probability should be checked first. For WebSphereMQ, the message throughput is neither affected by the filter complexity nor by the position of the component, which is decisive for the rejection of a message. As a consequence, we conclude that the filter logic of WebSphereMQ has a relatively high general filter overhead without optimization for complex AND-filters. This holds, since simple filter expressions take the same filtering effort as complex filter expressions, regardless of the early reject mechanism. Again, the ActiveMQ server outperforms all other servers.

3.3.3 Impact of IN-Filters (Presence Use-Case)

In the following experiment, we consider a more realistic scenario, the presence use-case as described in Section 2.3.2. It considers a practical scenario where a basic version of a presence information system is implemented. Typically, each user participating in such an environment can be identified by a unique ID. If the presence status of a user changes, a message will be sent containing the senders ID as application property. All entities interested in the presence information subscribe with a complex filter for the desired identifiers. For our experiment, we compare two design options:

(1) considering IN-filters, where IN-filters describe a set of IDs, and

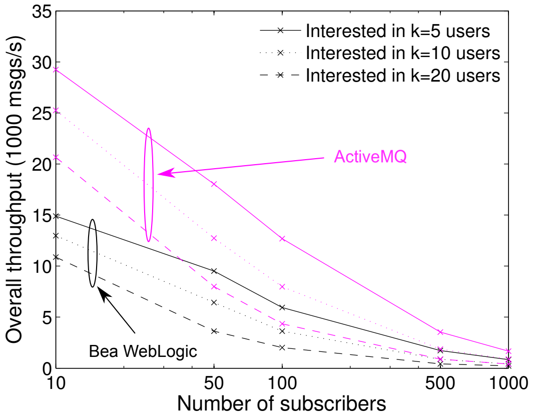
(2) a complex OR-filter searching for all IDs as complex filter components.

For this experiment, we set a constant replication grade of $r = 1$. We scale the number of active users from $m = 50$ up to $m = 1{,}000$. Each user is interested in the status of $k = 5; 10; 20$ other users including himself. The number of different IDs sent by the publishers is defined by $n_{ID} = m \cdot k$. Overall, 20 publishers send messages with application properties `ID`="val", where val $\in \{\text{“0000”}, \ldots, n_{ID} - 1\}$. If necessary, we prefix leading zeros to the value in order to get a string of constant length. We assume that each of the m users is interested in k specific IDs. In order to balance the experiment design, we ensure that user i is interested in the ID range $[i; (i + k - 1) \mod (n_{ID} - 1)]$. The experiment is conducted in two different ways by covering the interested ID range

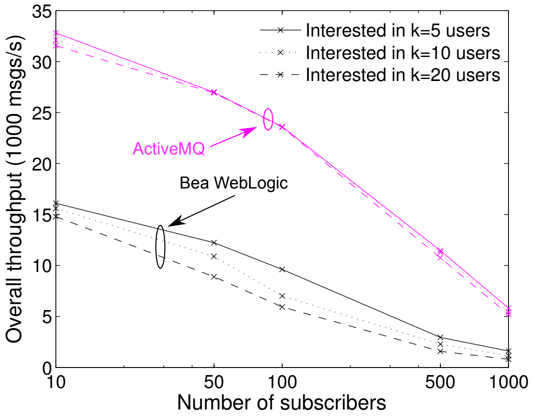
(1) an OR-filter, or

(2) an IN-filter

on one subscriber in order to emulate one user. The servers have to maintain at maximum $1{,}000$ TCP connections at the same time.



(a) Impact of OR-filters.



(b) Impact of IN-filters.

Figure 3.13: *Impact of the presence scenario on overall message throughput.*

The measurement results for Bea WebLogic and ActiveMQ in Figure 3.13 show that the overall message throughput performance of IN-filters is always better than the one observed for the corresponding OR-filters. A major decrease in the overall message throughput can be observed for the OR-filters when the number of filtered users k on the subscriber side increases. Whereas a larger set of IDs in the IN-filter scenario leads to a small throughput decrease for ActiveMQ, and a slight decrease of the throughput for Bea WebLogic. Our measurement results show that ActiveMQ outperforms the Bea WebLogic server by a factor of about two. In general, we can recommend to use IN-filters in such a presence use-case.

In Chapter 4 we introduce various models to predict the server throughput performance using complex OR and AND-filters. By adapting these models, a varying replication grade r can also be extrapolated for the IN-filters, respectively the presence scenario.

3.4 Impact of Subscription Aggregation and Registration

Besides the message throughput performance, we identified two other scenarios of interest, the time to register a set of registrations and the impact of subscription aggregation. Both scenarios are of interest for our experimental environment and in real appliances. The time to register connections covers typical flash crowd scenarios, like observed after a failure or during business hour. The aggregation of subscriptions might reduce programming overhead if a single machine has to maintain multiple virtual subscriptions at the same time.

3.4.1 Impact of Aggregation Options for Multiple Subscriptions

A test of the impact of different aggregation levels is performed in this section. The JMS API offers several connection types:

(1) Network connection

(2) JMS sessions

(3) Subscriptions

Each subscriber establishes a network connection, e.g., a TCP/IP connection to the JMS server. Several JMS sessions can be aggregated within such a network connection. A JMS session possibly contains multiple subscriptions installed by the client application, whereby a single subscription can hold at most one filter. When multiple subscriptions are set up between a subscriber machine and a server, a different number of network connections and JMS sessions can be used to support the same number of subscriptions. In the following experiment, the impact of different aggregation options are evaluated for $4{,}096$ subscriptions.

20 publisher threads are set up to send messages with string values from “0000” to “4095”. On the subscriber side, $4{,}096$ different subscriptions are set up, each of them having an application property filter for exactly one of the above numbers to assure a message replication grade of $r = 1$. $n_{\text{subscription}}$ different subscriptions are bundled into one JMS session and n_{session} different JMS sessions into one network connection of which n_{network} exist. Thus, a valid configuration must fulfill the equation $n_{\text{subscription}} \cdot n_{\text{session}} \cdot n_{\text{network}} = 4{,}096$. The overall server throughput is measured for different configurations.

Figure 3.14 shows the results depending on the number of JMS sessions for Bea WebLogic and ActiveMQ. We observe an overall message throughput for Bea WebLogic of about 800 msgs/s for all considered configurations. Hence, the aggregation options for multiple subscriptions have a rather small impact on the performance of Bea WebLogic. For ActiveMQ we observe a slightly different

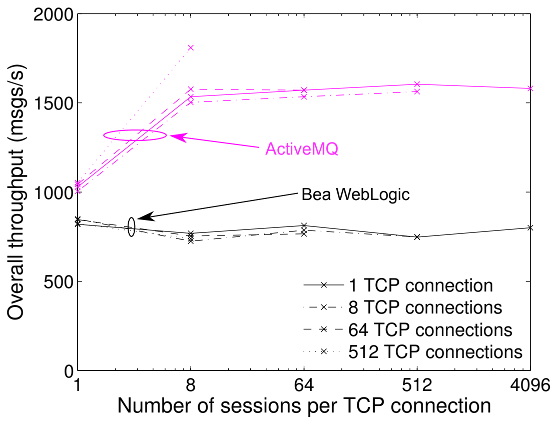


Figure 3.14: *Impact of the aggregation of 4096 subscriptions into a different number of TCP connections and JMS sessions on the server capacity.*

result. ActiveMQ benefits from grouping several JMS sessions. Using one JMS session per TCP connection, the achieved message throughput is noticeable lower than for 8 or more JMS sessions per TCP connection. This observation holds for all numbers of TCP connections. Furthermore, the highest throughput, with a throughput rate about 17% higher than the other configurations, is achieved with 512 TCP connections each carrying 8 JMS sessions. In general, we assume that the impact of the aggregation options, as also observed for Bea WebLogic, can be neglected in our experiments.

As a result for further experiments, we propose to establish for each JMS session a separate network connection and for each subscription a separate JMS session. If the number of subscriptions in the experiment is larger than the number of network connections supported by the server, several subscriptions are grouped into one JMS session.

3.4.2 Evaluation of the Registration Time for Subscriptions

In this section, we investigate the time duration which is needed to register different numbers of subscriptions. This is a crucial aspect since the registration of new subscriptions costs processing power. Furthermore, in case of flash crowds, i.e., of multiple simultaneous subscriptions, the response time of the server depends on the time needed to register the subscriptions. This is an important issue if we consider the so-called failover scenario described in [114]. In this scenario the message server, to which all clients are connected, fails and all clients have to reconnect to a backup server.

Scenario with Inactive Publishers

To observe the impact of the subscription process itself, we study first a scenario where subscribers register to the JMS server, but the publishers do not send messages. In general, the registration of a subscription is triggered at the subscriber by calling the synchronous *subscribe()* method. The subscription is successfully performed from the point of view of the subscriber by the time the method finishes. However, the JMS API states that it is not guaranteed that the subscription is already active on the server at that time. In the experiments, the registration of all subscriber threads is started simultaneously and we measure the time t_{reg} until the last thread returns from the *subscribe()* method call. Three different types of filters are considered in the experiments

(1) two simple filters,

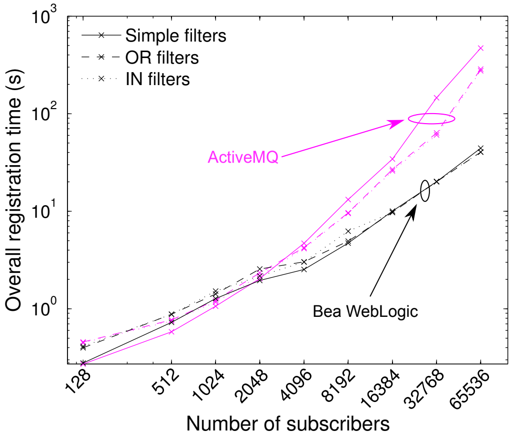
(2) OR-filters with two components, and

(3) IN-filters with two components.

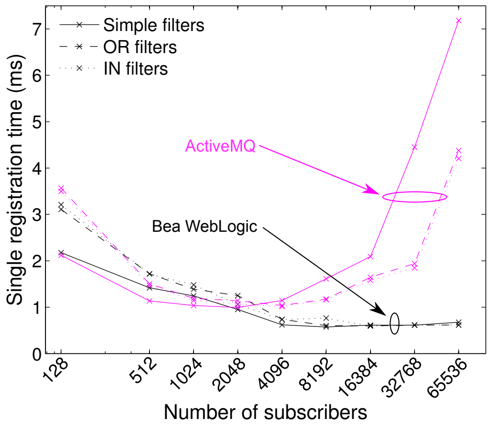
The experiment is performed for ActiveMQ and Bea WebLogic with $m \in \{128; 512, 1{,}024; 2{,}048; 4{,}096; 8{,}192; 16{,}384; 32{,}768; 65{,}536\}$ subscribers. The subscribers are run on up to 4 different machines to avoid a CPU bottleneck on any of the subscriber machines. For each filter type, the experiment was repeated five times. The results of the different runs are very similar such that the resulting confidence intervals are very small. However, they are omitted in the following figures for better readability.

Figure 3.15(a) shows the overall time to register all subscribers. The registration time scales almost linearly with the number of subscribers for Bea WebLogic. The overall subscription registration time for ActiveMQ server increases slightly more than linear. On the one hand, the graph shows that the registration of a large set of subscribers takes up to several seconds for Bea WebLogic and up to several minutes for ActiveMQ. On the other hand, a significant difference cannot be observed for the subscription times of different filter types.

Figure 3.15(b) illustrates the average time for the registration of a single subscription, i.e., $\frac{t_{\text{reg}}}{m}$. For Bea WebLogic this duration decreases with an increasing number of subscriptions and finally converges to a value of 0.6 ms. This result is counter-intuitive at first sight. However, for a small number of subscribers, a measurement overhead is observable. For a large number of subscribers the results are more reliable. Another reason might be, that it takes more time to install the first subscriptions. Until the internal data structures are large, the internal overhead increases to maintain the internal data structures. We can also observe the same behavior for ActiveMQ up to $4{,}096$ subscribers. With more simultaneous subscribers, the time to register a single subscription clearly increases again for the ActiveMQ server.



(a) Overall time for the registration of all subscribers.



(b) Average time for the registration of a single subscriber.

Figure 3.15: *Time to register simultaneously starting subscriptions with inactive publishers.*

Thus, only for Bea WebLogic we can notice that the registration time for a filter does not increase with the number of already existing subscriptions. Again, no significant difference is observed among different filter types, except the impact of simple filters on the registration time for ActiveMQ.

Scenario with Active Publishers

We perform a similar experiment with active publishers, i.e., publishers sending messages that are not matched by already registered subscriptions. Hence we set a replication grade of $r = 0$. This works well with the ActiveMQ server, but the experiment cannot be conducted with the BEA WebLogic JMS server: the server server stops accepting new subscriptions and blocks publishers from sending and never returns to its normal operation without displaying any failure notifications. Thus, we focus on the results for the ActiveMQ server.

Figure 3.16 shows a different representation of the measured data. The figure considers on the x-axis the experiment run time and depicts on the y-axis the observed received message throughput. During a single measurement run, one of $m \in \{128; 512; 1{,}024; 2{,}048; 4{,}096; 8{,}192; 16{,}384\}$ subscriptions with a simple filter are connected to the ActiveMQ server. The figure shows clearly the warm-up phase and the stable throughput conditions before the experiment starts. After this point in time the received throughput clearly decreases, but the shape remains remarkably constant for all numbers of m. If the desired number of subscriptions is reached, we can observe a slight impact of internal rearrangements of the JMS server, while it eventually converges to a certain level of received message throughput. Several repetitions of the same experiment run lead to the same result, so we again can omit the confidence intervals. We have to keep in mind that the y-axis is scaled logarithmic, so the stable phase of $m = 16{,}384$ is still reached at about 200 msgs/s.

With about 230 s for installing $m = 8{,}192$ subscribers, the scenario with active publishers is an order of magnitude slower than the one with inactive publishers, where it took about 10 s to connect.

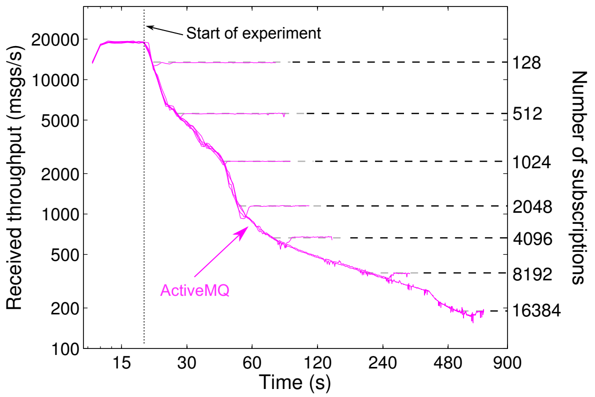


Figure 3.16: *Impact of number of parallel subscriptions on received throughput.*

The time to register a single subscription is in the order of milliseconds. As a consequence, the registration of large set of subscriptions may take several minutes. This is a critical issue for failover cases.

Evaluation of the First Message Delay

In a realistic scenario, the currently connected subscribers also receive messages, while new subscription registrations arrive at the JMS server. The experiment conducted in this section considers a constant rate of new subscription registration arrivals while dispatching messages to the already connected subscribers. We monitor the delay t_{tts} between the start of the registration request and the finished subscription. A subscriber is registered, when the *subscribe()* API method execution is completed. Additionally, we monitor the delay t_{fma} between completion of the *subscribe* API method and the time until the first message arrives at the newly registered subscriber. There are different ways a JMS server can handle new subscriptions. It can accept a new subscription and return as fast as possible to block the subscriber for a minimum amount of time. This might lead to an internal queue of registration requests, where we assume that t_{fma} differs from t_{tts}. Otherwise, the subscriber can block until all internal data structures are fully updated, which becomes difficult in distributed JMS server environments. Hence, the execution of the *subscribe()* API method does only guarantee that the connection between the subscriber and the JMS server is established. But a subscribers registration process at the JMS server cannot be assumed as successfully completed until the first successful message delivery. The delay between a successful return of the API method and the first message is a crucial aspect, since the subscriber assumes it is connected, but in reality it experiences message loss.

In order to quantify the delay between t_{tts} and t_{fma} we designed an experiment considering realistic conditions. We connect 20 publishers to the ActiveMQ server. Each publisher sends messages in saturated mode with a unique application property set and a property to identify the sending publisher. Thus, we assume that at any point in time, a message is available on the JMS server for dispatching, by using internal monitoring capabilities of the JMS server. Furthermore, we can measure which messages are available for dispatching on the JMS server for a new arriving registration. This allows us to calculate the number of lost messages. We connect up to 320 subscribers, each using a simple filter for a unique application property. In order to balance the experiment and to avoid discarding messages, each unique application property is filtered by up to 16 subscribers. Also the subscribers register for the unique properties in a round robin fashion. The inter-arrival time of the registrations is set to a constant delay of 2 seconds, which is larger than the maximum observed single registration time.

In Figure 3.17 the *complementary cumulative distribution function* (CCDF) of the observed single registration delay times $t_{\text{reg}}^{\text{sub}}$ is plotted. The single registration time t_{tts} increases for the subscribers exponentially. The delay to the first message arrival t_{fma} increases similar to the single registration time. Overall, more than 10% of the registrations take longer than 500 ms. The delay between the successful registration and the first message has an average of 56 ms for our

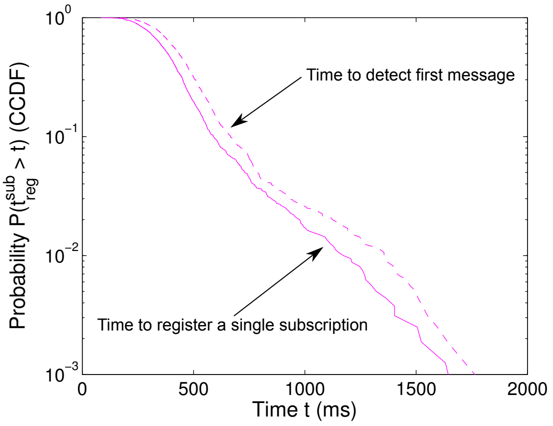


Figure 3.17: *ActiveMQ: Delay between a single registration and detecting the first message.*

scenario. During this period of time we measured an average loss of about 100 messages in our saturated scenario. We can conclude that an application considering a high rate of registrations has to consider lost messages or to establish a synchronization mechanism between the communicating partners, especially for a failover scenario and in distributed JMS server environments.

3.5 Concluding Remarks

In the preceding chapter, we measured and evaluated the message throughput of the FioranoMQ, SunMQ, WebSphereMQ, Bea WebLogic, and ActiveMQ JMS servers under various conditions. The introduced testbed and the measurement methodology enables a reliable and controllable measurement based evaluation of all kinds of servers. We also present a set of basic measurements, i.e., the client scalability, data and message throughput, and message selection to identify the impact on the measured middleware environment and their performance limits. The enhanced measurements show that the throughput clearly depends on the replication grade and the number of filters and their complexity. The measured throughput performance for the five investigated server types spans over several orders of magnitude.

The next chapter introduces a rather complex experiment series based on the results of the experiments presented in this chapter. The goal is to evaluate the joint impact of filters and the message replication grade by measurement and to propose mathematical approximation models of the message processing time for each server type. Using these models, we can predict the message throughput for specific application scenarios and can omit dedicated measurements.

4 Evaluating Message-Oriented Middleware

In natural sciences it is common to observe and measure physical systems and to describe the results with an abstract model. Depending on the system this model can be very complex. If we consider software packages, the number of implemented methods, features, and their interdependencies may have a major influence on the overall performance of the software package.

Our approach to retrieve a rough model for system performance throughput estimation is divided into the following major steps:

1. Design and run a set of experiments, which supports the understanding of the system.

2. Based on the results of this experiment, apply a model which follows the set of chosen parameters.

3. Calculate the system specific parameter set by multiple regression and least squares approximation.

4. Validate the model by applying the calculated system values using additional measurement runs.

Using this approach we are able to find a rough estimating model for all JMS servers. We can even extend the models in order to cover additional aspects by varying the design parameters of the experiments.

We know from the filter activation experiment in Section 3.2.5 and the complex filter experiments in Sections 3.3.1 and 3.3.2 that different numbers of installed filters and the replication grade have a major impact on the server capacity. Therefore, we start with an experiment series set up such that we can study the joint impact of filters and replication grade on the message throughput. As a result, we propose a simple analytical model to describe the dependencies and fit the model parameters to the measurement data.

Furthermore, the impact of different filtered properties such as correlation ID filters and application property filters are evaluated. For the SunMQ and the WebSphereMQ servers an adapted model is proposed, since their behavior cannot be modeled using our basic assumption. In addition, for the ActiveMQ server, we present an extended model, which considers not only different filter types and but also lengths to the simple filters evaluated for the other servers.

This chapter starts with a general introduction to multiple regression and least-squares approximation in Section 4.1. In Section 4.2, a basic model is introduced and evaluated for several JMS servers. Since this model is not valid for all servers, we show some adapted models for the WebSphereMQ and the SunMQ server in Section 4.3. Due to the fact that the first experiments do not consider a complex filter scenario, we enhance our basic model and validate it for the ActiveMQ server in Section 4.4. The concluding section presents an application of the evaluated models and some remarks on the results.

4.1 Background: Multiple Regression and Least-Squares Approximation

Linear regression is a method of modeling a dependent variable Y as a function of a single variable x and is used in our approach to calculate the system specific parameters, which characterize the performance of a certain JMS server. In systems with increased complexity we have to use multiple variables $x_1, \ldots, x_n$. If these variables are independent and the response on each variable can be modeled by a linear influence, the dependent variable Y can be written more generally as a set of i functions considering n independent variables

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + +\beta_n x_{i,n} + \epsilon_i, \tag{4.1}$$

where the error ϵ_i is a variable with mean zero and variance σ^2. The mean

$$\mathrm{E}\left[Y_i\right] = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \ldots + \beta_n x_{i,n} \tag{4.2}$$

of the multiple linear regression model is defined such that it is also a linear function of regression parameters $\beta_0, \beta_1, \ldots, \beta_n$.

To fit the model, we associate a least-squares function derived from Equation (4.1), which has to be minimized with respect to $\beta_0, \beta_1, \ldots, \beta_n$:

$$L = \sum_{k=1}^{i} \epsilon_k^2 = \sum_{k=1}^{i} \left(y_k - \beta_0 - \sum_{j=1}^{n} \beta_j x_{kj} \right)^2 \tag{4.3}$$

Using multiple regression models, it is convenient to use the related variables and operations in matrix notation. Hence, Equation (4.1) can be transformed to

$$Y = X\beta + \epsilon. \tag{4.4}$$

In general, Y is an $(i \times 1)$ vector of the observations, X is an $(i \times n)$ matrix of the levels of the independent variables, β is a $(n \times 1)$ vector of the regression coefficients, and ϵ is an $(i \times 1)$ vector of random errors:

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \end{pmatrix} \quad X = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{i1} & x_{i2} & \dots & x_{in} \end{pmatrix}$$

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{pmatrix} \qquad \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_i \end{pmatrix}.$$

The vector of least-squares estimators is searched, which minimizes

$$L = \sum_{k=1}^{i} \epsilon_k^2 = \epsilon' \epsilon = (y - X\beta)'(y - X\beta). \tag{4.5}$$

The problem stated in Equation (4.5) can be reformulated using the Euclidean vector norm. The Euclidean vector norm for vector x is defined as

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}. \tag{4.6}$$

With the Euclidean vector norm, the minimization problem to be solved using Equation (4.5), is

$$\min_{\beta} \|X\beta - y\|_2 . \tag{4.7}$$

One problem with the solutions provided by the above described approach is that negative values might be valid solutions and represent a “best fit”. In statistics, the problem can be divided in several classes of linear least-squares problems that have to meet additional inequality constraints. A very common class consists of those with non-negativity constraints. In this work, we calculate overhead times, which means that the regression coefficients, by definition, can only have non-negative values. This problem is known as the *non-negative least-squares* (NNLS) problem and can be formulated as

$$\min_{\beta} \|X\beta - y\|_2 \quad (\beta \geq 0). \tag{4.8}$$

The Nonnegative Least Squares problem is a standard problem in numerical linear algebra [111], [128]. A number of commercial [147], [148], [142], and open source libraries [64] provide approximation algorithms to solve the problem.

In our work, we use the LSQNONNEG function of the MATLAB software suite [142]. The algorithm starts with a set of possible basis vectors and computes the associated dual vector λ. It then selects the basis vector corresponding to the maximum value in λ in order to swap out of the basis in exchange for another possible candidate. This continues until $\lambda \leq 0$. LSQNONNEG uses the algorithm described in MATLAB documentation [128].

The least-squares approximation is only one possible statistical method for fitting. The method of maximum likelihood can be used as well. However, it can be shown that the least squares estimates of the regression parameters $\beta_0, \beta_1, \ldots, \beta_i$ are maximum likelihood estimates. Thus, we decided to calculate only the least-squares approximation.

4.2 Modeling the Server Capacity

In the following sections, we present and evaluate performance models for the servers measured in Chapter 3. In general, the section is structured as follows: We describe the experiment series for the desired server, suggest a suitable mathematical approximation model for the server throughput, and fit the corresponding model parameters. The evaluation is concluded by a validation of the proposed model.

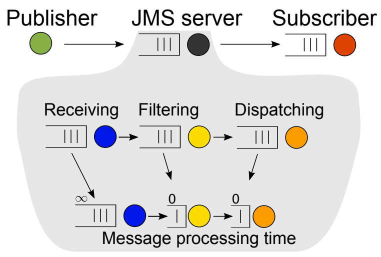


Figure 4.1: *Motivation for the modeling process.*

4.2.1 Performance Model for the Message Processing Time

As identified in Chapter 3, the maximum message throughput of a JMS server can be used as a measure. Thus, it is interesting to model this measure to design or dimension a real-world system. Our goal is to present a set of key performance indicators for a JMS server in combination with an analytical model. This approach reduces the effort for dimensioning the performance of new hardware and application scenarios, by reducing the necessary measurement points for calibration. The focus on certain critical parameters, which can be adapted to the application scenario, differs our approach from pure benchmarking. Therefore, the following assumptions have to be made.

We assume first, based on the results of our measurements, that the processing time of the JMS server for a message consists of three components. As depicted in Figure 4.1, the overall system consists of three parties, the publisher, the subscriber, and the JMS server itself as relaying element. The internal processing of the server can be divided into three partitions. On message arrival, the server has to move this message to an internal queue. Another process proceeds with the message filtering task. After the filter evaluation, the server has to dispatch each message to the desired subscriber, which might also be seen as a separate task. The experiences from the measurements show that the message processing can

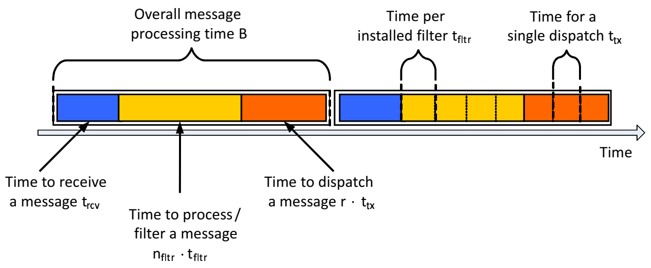

Overall message processing time B
Time per installed filter t_{fltr}
Time for a single dispatch t_{tx}
Time
Time to receive a message t_{rcv}
Time to process / filter a message $n_{fltr} \cdot t_{fltr}$
Time to dispatch a message $r \cdot t_{tx}$

Figure 4.2: *Basic model for the system utilization at the JMS server.*

be assumed as an independent process for each message. Thus, we can aggregate the waiting space. This leads to an overhead time model as depicted in Figure 4.2. For each processed message, there is

- a fixed time t_{rcv} which is almost independent of the number of installed filters for a constant message size.
- a fixed time t_{fltr}, which the JMS server needs to evaluate if a filter matches a message. If there are n_{fltr} filters installed on the server, the time needed to match all filters is $n_{\mathrm{fltr}} \cdot t_{\mathrm{fltr}}$. This value depends on the application scenario.
- a fixed time t_{tx} to forward a message. It depends on the message replication grade r. The time to forward a message to all recipients is $r \cdot t_{\mathrm{tx}}$, which corresponds to the time the server needs to forward r copies of the message.

Combining the described parameters leads to the following message processing time B:

$$B = t_{\mathrm{rcv}} + n_{\mathrm{fltr}} \cdot t_{\mathrm{fltr}} + r \cdot t_{\mathrm{tx}}. \tag{4.9}$$

Within time B, one message is received and r messages are sent on average.

Therefore, the received and overall throughput can be calculated by $\frac{1}{B}$ and $\frac{r+1}{B}$, respectively.

We call the model in Equation (4.9) our "base model". Not all servers follow the linear scale in the same way as our base model proposes. In the next section, we first evaluate a separate validation of the model for the FioranoMQ, the Bea WebLogic, and the ActiveMQ JMS server, where the base model applies.

4.2.2 FioranoMQ

The observations from the FioranoMQ server inspired major parts of the presented model. Thus, we start our evaluation and validation of the previously introduced model using this server. Different measurement series for calculating the parameters and verifying the model are conducted. The following section starts with a description of the measurement series setup and results gained from the measurement runs. The results are validated in a separate section.

4.2.2.1 Experiment Setup and Measurement Results

The overall testbed setup includes three measurement machines, one as a publisher, one as a subscriber emulating machine, and a dedicated FioranoMQ server machine. The following experiments include experiments for correlation ID and application property filtering. Both parameters are part of the JMS message header, where the correlation ID is part of the fixed header and the application property is part of the user-defined header area. We expect that the correlation ID filtering is close to the maximum achievable performance by a specific server, whereas the application properties will represent the more realistic scenario. A detailed introduction to the JMS header is given in Section 2.2.

Five publishers are connected to the JMS server and send messages with correlation ID #0 or application property value #0 in a saturated way. Furthermore, $m_{\text{subs}}^{\text{add}}+r$ subscribers are connected to the JMS server, r of them filter for application property value #0, while the other $m_{\text{subs}}^{\text{add}}$ subscribers filter for value

#1. Hence, $m_{\text{subs}}^{\text{add}}+r$ 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 $m_{\text{subs}}^{\text{add}} \in \{5; 10; 20; 40; 80; 160\}$ additional subscribers. The scenario with $n = 0$ is covered by the filter activation experiment.

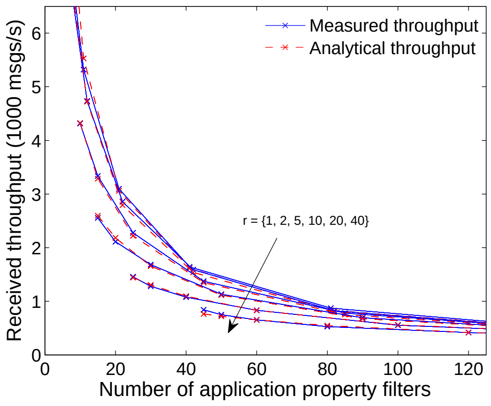
Table 4.1: *FioranoMQ: Empirical values for the model parameters of the message processing time in Equation (4.9)*

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

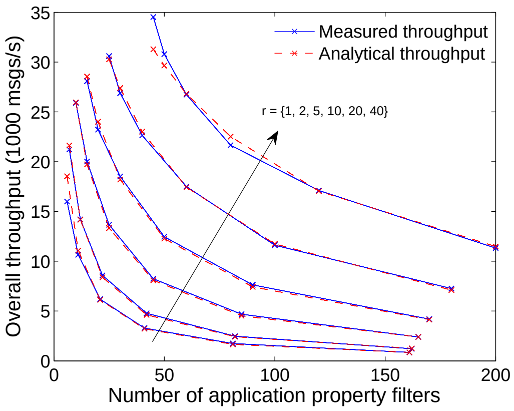
Figures 4.3(a) and 4.3(b) show the received and overall message throughput for application property filters depending on the number of installed filters $n_{\text{fltr}} = m_{\text{subs}}^{\text{add}} + r$ and on the replication grade r. The solid lines depict the measured throughput. An increasing number of installed filters obviously reduces the message throughput of the server. An increasing replication grade decreases the received message throughput, but it increases the overall message throughput of the server to a certain extent. We obtain similar measurement curves with about 100% more throughput for correlation ID filters. Since the shape of the curves is identical we skipped the corresponding figures. In addition, we conduct the same experiment series with the $m_{\text{subs}}^{\text{add}}$ non-matching filters set to #1, . . . , #$m_{\text{subs}}^{\text{add}}$. They lead to the exactly same results as in Figures 4.3(a) and 4.3(b). Thus, we cannot find any throughput improvement if the same filters are used instead of different filters. This implies that the model can also be used to predict the performance in scenarios where different filters are used, which might be more realistic.

4.2.2.2 Validation of the Model by Measurement Data

The results in Figures 4.3(a) and 4.3(b) visualize the received and overall throughput. Within time B, one message is received and r messages are dispatched by the server. Thus, the overall throughput is given by $\frac{r+1}{B}$ and corresponds to the measurement results in Figure 4.3(b).



(a) Received message throughput – measurements and analytical data.



(b) Overall message throughput – measurements and analytical data.

Figure 4.3: *FioranoMQ: Impact of the number of filters* n_{fltr} *and the message replication grade* r.

The parameters n_{fltr} regarding the number of installed filters and r regarding the replication grade used for the message processing time in Equation (4.9) are known from the respective experiments. We fit the parameters t_{rcv}, t_{fltr}, and t_{tx} by the least squares approximation described in Section 4.1 to adapt the linear model in Equation (4.9) to the measurement results. The resulting parameter values are compiled in Table 4.1 for correlation ID and application property filters. The time t_{fltr} to filter a message is of an order of magnitude faster for correlation ID filtering than processing application property filters. Since correlation IDs are part of the fixed header, the internal filtering can process these kind of messages more efficiently. This behavior might also lead to an improved message receiving time t_{rcv}, whereas the time to dispatch a message t_{tx} remains constant. The improvement of the t_{rcv} by an order of magnitude is also caused by the multiple regression method, which tends to prefer the first regression parameter. But this does not affect the quality of our performance evaluation.

We calculate the message throughput based on these values and Equation (4.9) for all measured data points, and plot the results with dashed lines in Figures 4.3(a) and 4.3(b). The throughput from our analytical model fits very well with our measurements for all numbers of filters n_{fltr} and all replication grades r, considering the result as a rough estimation of the overall system performance. Additionally measured data points in the defined range, which are not considered in the regression process, enabled a double check of the quality of the model.

4.2.3 BEA WebLogic Server

The server configuration for the following validation of the Bea WebLogic is the same as described in Section 3.1.2.

4.2.3.1 Experiment Setup and Measurement Results

We set up 20 publishers on a single machine, which send only messages with ID #0 to the server. According to our observations in the experimental chapter, this increased number of publishers is necessary to saturate the JMS server. To achieve a message replication grade of r, we set up r subscribers with a filter for ID #0. Furthermore, we install $m_{\text{subs}}^{\text{add}} \in \{5; 10; 20; 40; 80; 160\}$ additional subscribers with a filter for ID #1. Thus, the overall number of subscribers or filters on the server is $n_{\text{fltr}} = m_{\text{subs}}^{\text{add}} + r$. We vary this number of subscribers equally distributed over two subscriber machines. Since one physical machine might present a bottleneck, we increased the number of subscriber machines.

Figures 4.4(a) and 4.4(b) show the results for complex application property filters for different settings of n_{fltr} and r. Both the received and the overall throughput slightly decrease with an increasing number of filters. However, the message replication grade r has a larger influence on the throughput. Again, an increasing replication grade decreases the received throughput while it increases the overall throughput.

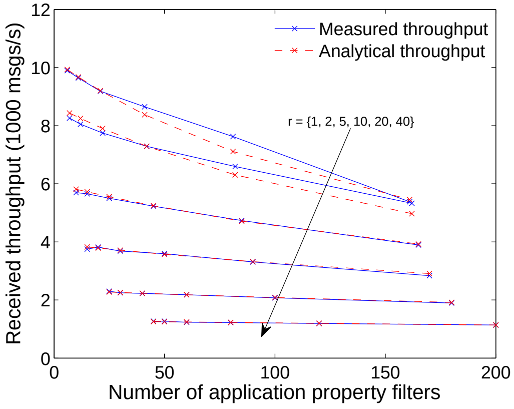
We also conduct another experiment where the $m_{\text{subs}}^{\text{add}}$ additional filters are different, but we obtain exactly the same results as in Figures 4.4(a) and 4.4(b). Hence, the BEA WebLogic JMS server does not take advantage of same filters in the system. Furthermore, we perform the same experiment series for correlation ID filters and the results showed a slightly larger throughput, but the throughput curves are qualitatively similar. Therefore, we omitted the figures.

4.2.3.2 Validation of the Model

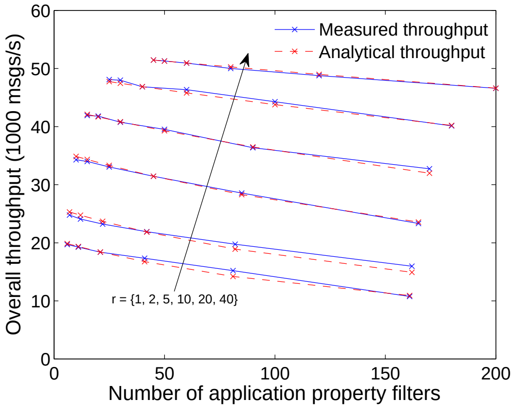
Again, we derive the values for t_{rcv}, t_{fltr}, and t_{tx} by the least-squares approximation based on the model in Equation (4.9) and the experimental results for the received throughput for all parameter combinations of n_{fltr} and r. Table 4.2 shows the obtained model parameters for both application property and correlation ID filters.

Table 4.2: *Bea WebLogic: Empirical values for the model in Equation (4.9).*

parameter	t_{rcv} (s)	t_{fltr} (s)	t_{tx} (s)
corr. ID filtering	$7.6239 \cdot 10^{-5}$	$3.1410 \cdot 10^{-7}$	$1.6944 \cdot 10^{-5}$
app. prop. filtering	$8.0182 \cdot 10^{-5}$	$5.3332 \cdot 10^{-7}$	$1.7319 \cdot 10^{-5}$



(a) Received message throughput – measurements and analytical data.



(b) Overall message throughput – measurements and analytical data.

Figure 4.4: *Bea WebLogic: Joint impact of the number of installed filters and the replication grade on the message throughput for application property filters.*

We use these parameters to analytically calculate the throughput for the same parameters sets as in the experiments in Section 4.2.3.1 and draw the results as dashed lines in Figure 4.4(a) and Figure 4.4(b). The analytical throughput is similar to the measured data for the evaluated parameter range. Looking at the received throughput, the model tends to underestimate the number of received messages for lower values of r, but is still a good estimator of the overall system performance.

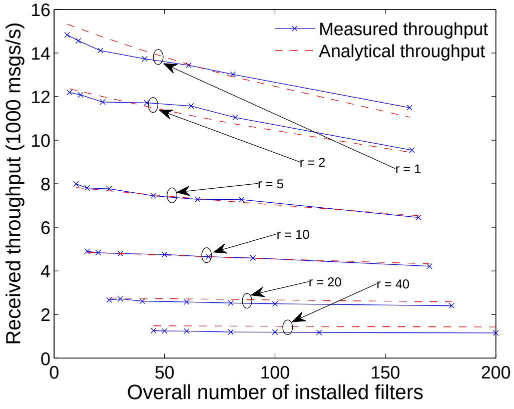
4.2.4 Apache ActiveMQ

The configuration of the ActiveMQ server is also used without any modifications, as described previously in the experimental chapter.

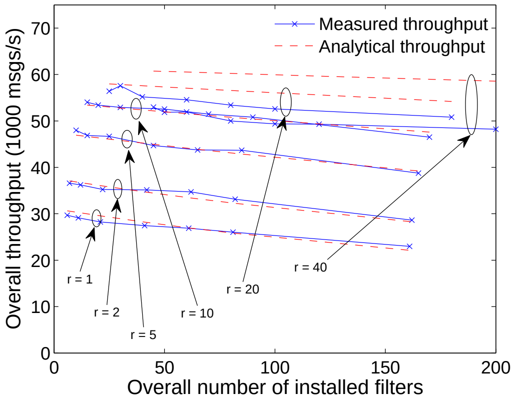
4.2.4.1 Experiment Setup and Measurement Results

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, which filter for ID #0. We used the same values for r, n_{fltr}, and $m_{\text{subs}}^{\text{add}}$ as in the previous experiments. The measurement runs are conducted with 20 publisher threads on one publisher machine and with a variable number of $r+m_{\text{subs}}^{\text{add}}$ subscribers equally distributed over two subscriber machines.

The solid lines in Figure 4.5(a) and Figure 4.5(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. To illustrate this effect, we provide in Figure 4.6(a) and Figure 4.6(b) 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 4.6(a) and Figure 4.6(b) show that the received throughput decreases and the overall throughput increases with an increasing replication grade. Comparing Figure 4.5(a) and Figure 4.5(b) with Figure 4.6(a) and Figure 4.6(b) leads to the conclusion that the impact of the message replication grade on the message throughput is larger than the impact of the number of installed filters.



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



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

Figure 4.5: *ActiveMQ: Joint impact of the number of installed filters and the replication grade on the message throughput for application property filters.*

The before observed effect becomes obvious when the replication grade increases to $r = \{20, 40\}$. At this point the observed overall throughput decreases again, which might be caused by another system bottleneck. We cannot specify the bottleneck with the considered parameters. Since the CPU utilization in all scenarios is close to 100% there might be another I/O bottleneck, e.g., the hard disk performance.

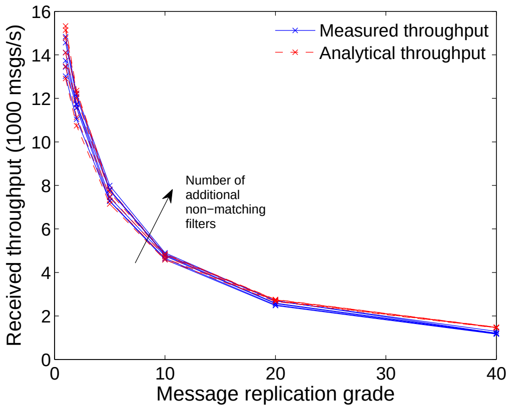
Table 4.3: *ActiveMQ: Empirical values for the model parameters of the message processing time in Equation (4.9).*

parameter	t_{rcv} (s)	t_{fltr} (s)	t_{tx} (s)
corr. ID filtering	$4.58 \cdot 10^{-5}$	$1.46 \cdot 10^{-7}$	$1.64 \cdot 10^{-5}$
app. prop. filtering	$4.88 \cdot 10^{-5}$	$1.62 \cdot 10^{-7}$	$1.54 \cdot 10^{-5}$

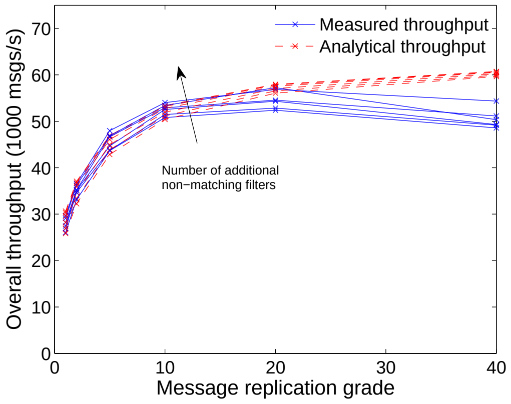
The throughput is the same, regardless if we use the same or different filters that do not match. For this server implementation each filter is evaluated, without considering that the same filter has to be evaluated multiple times. The same experiment for correlation ID filters leads to very similar results and therefore we omitted showing the results in detail.

4.2.4.2 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 consider only the experiments with replication grades $r \in \{1; 2; 5; 10\}$ in the least squares approximation and obtain the model parameters printed in Table 4.3. We use these parameters to calculate the analytical throughput which is illustrated in Figures 4.5 and 4.6 by dashed lines. For small replication grades $r = \{1; \ldots; 10\}$ the analytical throughput is in good accordance with the measured throughput. We also evaluated the performance of the correlation ID filters. However, we measured only slightly different values for the message throughput and we can therefore omit a rather complex model.



(a) Impact of the replication grade on the received throughput.



(b) Impact of the replication grade on the overall throughput.

Figure 4.6: *ActiveMQ: Impact of the replication grade r and the non-matching additional filters on the throughput.*

As mentioned above, the capacity curves for message replication grades $r = 20$ and $r = 40$ in Figure 4.5(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. For the FioranoMQ and the Bea WebLogic server we do not encounter 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 the FioranoMQ and the Bea WebLogic was sufficient even for a large message replication grade of $r = 40$. However, we expect to observe similar saturation effects for all servers, if we further increase the message replication grade in this experiment series.

Besides the observation that another parameter might limit the modeling capabilities, as seen for the ActiveMQ, there are other server implementations, which have a completely different internal message processing strategy. The next chapter introduces adapted models for the SunMQ and the WebSphereMQ server.

4.3 Adapted Performance Models

For the SunMQ and the WebSphereMQ server, the basic model does not apply. Several measurements showed a clearly different behavior. Therefore, we decided to evaluate separate models for the two server types. In general, this approach shows the ability to enhance our procedure to unknown or maybe updated server behaviors. This is also a clear difference to basic benchmarking, since our approach supports to understand and identify possible system bottlenecks, whereas pure benchmarking leads only to a comparative measure.

4.3.1 SunMQ: Increased Impact of Different Filters

From our previous measurements we observed some kind of filter optimization behavior. In order to verify and evaluate this behavior we adapted the experiment design and the model for SunMQ, which leads to an increased complexity in the model. The configuration for the SunMQ server is described in Section 3.1.2.

4.3.1.1 Experiment Setup and Measurement Results

First, we performed the same experiment series as in the previous sections for SunMQ and found out that it matters whether non-matching filters are the same or different. Thus, we redesign the experiment series in such a way that we can study the impact of the replication grade r, the number of different filters $n_{\text{fltr}}^{\text{diff}}$, and the number of overall installed filters $n_{\text{fltr}}^{\text{all}}$ on the message throughput. The publishers send only messages with value #0. To achieve a replication grade of r, we set up r subscribers with a filter for value #0. Furthermore, we install $n_{\text{diff}}^{\text{add}}$ other different filters for values from #1 to #$n_{\text{diff}}^{\text{add}}$. We set up these additional filters f_r times and call f_r the filter replication factor in this experiment. For our experiments we use the following values for $r \in \{1;2;5;10;20;40\}$, $n_{\text{diff}}^{\text{add}} \in \{1;2;5;10;20;40;80;160\}$, and $f_r \in \{1;2;4;8\}$. Overall we installed 5 publisher threads and varied the number of $r + (n_{\text{diff}}^{\text{add}} \cdot f_r)$ subscriber threads accordingly.

Figures 4.7 and 4.8 show the received and overall message throughput for this experiment series. The server capacity clearly decreases for an increasing number of different filters $n_{\text{diff}}^{\text{add}}$. An increasing message replication grade r reduces the received message rate, but it increases the overall message rate. The four related figures differ with a varying filter replication grade f_r, but they look very similar at the first spot. The impact of the number of all filters $n_{\text{fltr}}^{\text{all}} = r + (n_{\text{diff}}^{\text{add}} \cdot f_r)$ is clearly visible when we compare the right margins of the figures since the number of all filters only differs significantly if the number of additional different filters $n_{\text{diff}}^{\text{add}}$ is large. Thereby, we observe that using the same filters also reduces the throughput even though they do not match.

4.3.1.2 Modeling the Message Processing Time

The message processing time is the inverse of the received message throughput. Figure 4.7 shows that it depends on the number of additional filters $n_{\text{fltr}}^{\text{add}}$, the filter replication factor f_r, and the replication grade r. We propose a simple model for the message processing time B that relies on all filters $n_{\text{fltr}}^{\text{all}} = r + (n_{\text{diff}}^{\text{add}} \cdot f_r)$ and the number of different filters $n_{\text{fltr}}^{\text{diff}} = n_{\text{diff}}^{\text{add}} + 1$:

$$B \quad = \quad t_{\text{rcv}} + n_{\text{fltr}}^{\text{all}} \cdot t_{\text{fltr}}^{\text{all}} + n_{\text{fltr}}^{\text{diff}} \cdot t_{\text{fltr}}^{\text{diff}} + r \cdot t_{\text{tx}}. \tag{4.10}$$

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

4.3.1.3 Validation of the Model by Measurement Data

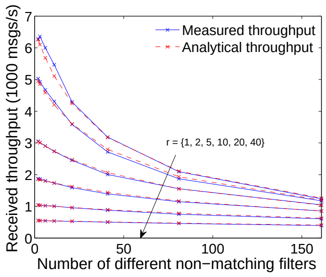
The results in Figure 4.7 show the overall throughput regarding received and sent messages. The parameters $n_{\text{fltr}}^{\text{diff}}$, $n_{\text{fltr}}^{\text{all}}$, and r for the message processing time B are known from the respective experiments. Again, we can fit the parameters t_{rcv}, $t_{\text{fltr}}^{\text{all}}$, $t_{\text{fltr}}^{\text{diff}}$, and t_{tx} by the least squares approximation to adapt the model in Equation (4.10) to the measurement results. The results are listed in Table 4.4 for application property filters.

Table 4.4: *SunMQ: Empirical values for the model parameters of the message processing time in Equation (4.10).*

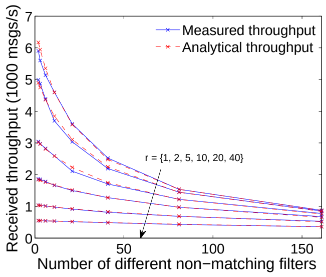
parameter	t_{rcv} (s)	$t_{\text{fltr}}^{\text{all}}$ (s)	$t_{\text{fltr}}^{\text{diff}}$ (s)	t_{tx} (s)
app. prop. filtering	$1.12 \cdot 10^{-4}$	$2.20 \cdot 10^{-6}$	$1.76 \cdot 10^{-6}$	$4.01 \cdot 10^{-5}$

We calculate the message throughput based on these values and Equation (4.10) for all measured data points, and plot the results with dashed lines in Figures 4.7 and 4.8. With the proposed adaption in the model the throughput from our analytical model fits well with our measurement results.

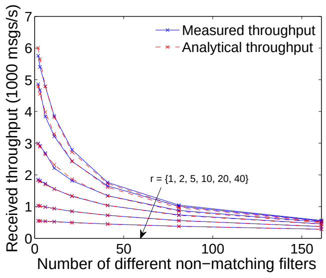
Thus, only the extra effort for different filters differs from the basic model. So we assume that the main difference between the servers following the basic model and the SunMQ is the internal filter evaluation strategy.



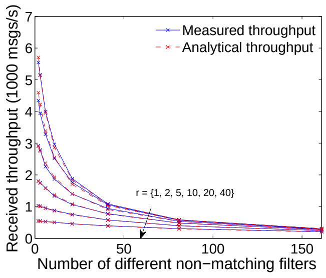
(a) Filter replication grade $f_r = 1$.



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

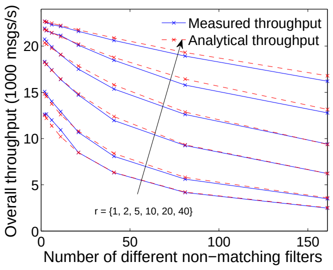


(c) Filter replication grade $f_r = 4$.

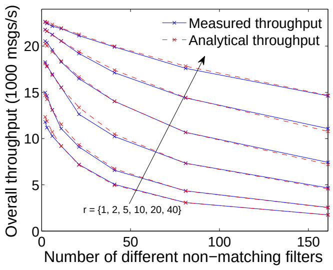


(d) Filter replication grade $f_r = 8$.

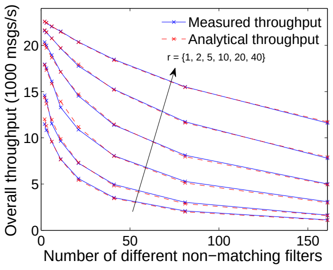
Figure 4.7: *SunMQ: Impact of the number of different filters n_{fltr}^{diff} and the message replication grade r on the received message throughput for different numbers of additional identical filters.*



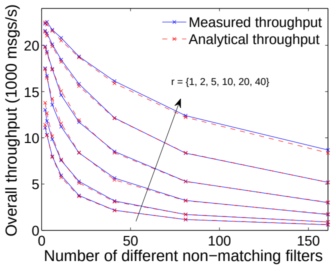
(a) Filter replication grade $f_r = 1$.



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



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



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

Figure 4.8: *SunMQ: Impact of the number of different filters $n_{\mathit{fltr}}^{\mathit{diff}}$ and the message replication grade r on the overall message throughput for different numbers of additional identical filters.*

4.3.2 WebSphereMQ: Impact of Filtering Dominates Dispatching

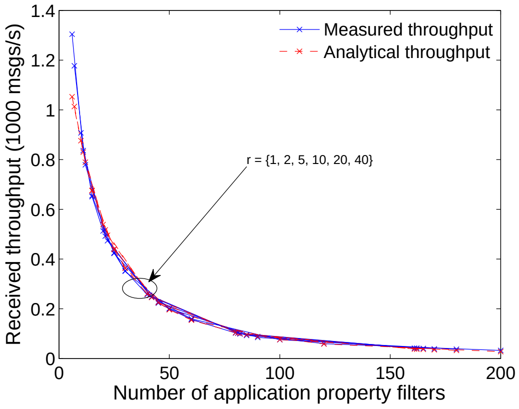
The WebSphereMQ requires a substantially different model for the message processing time compared to the basic model, e.g., applied to FioranoMQ, and the one presented for SunMQ.

4.3.2.1 Experiment Setup and Measurement Results

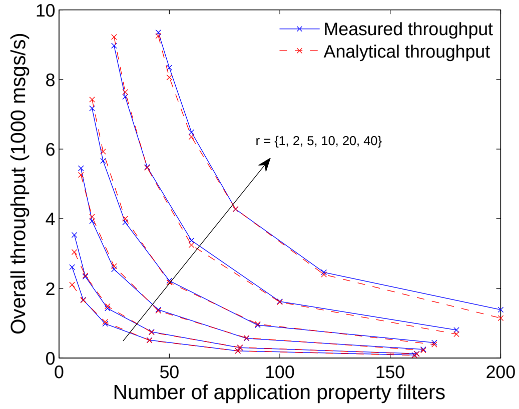
We set up the same series of experiments as for the FioranoMQ in Section 4.2.2.1. Figure 4.9(a) shows the received message throughput depending on the number of installed filters $n_{\text{fltr}} = m_{\text{subs}}^{\text{add}} + r$ and on the replication grade r. The solid lines show the measured throughput. An increasing number of filters reduces the received message throughput of the system which is independent of the replication grade for the considered scenarios. This is different to the results found for the servers following the basic model in Section 4.2.1, like for FioranoMQ. It is even different from the one applied for SunMQ in Section 4.3.1.1. We assume, that this observation results from a dominating filter processing time in comparison to the time to transmit a message. Figure 4.9(b) shows the resulting overall message throughput. It decreases also with an increasing number of filters, but it rises with the replication grade. We have performed the same experiments for correlation ID filters, too, and obtained the same measurement results. Thus, correlation ID and application property filters lead to the same throughput both for SunMQ and WebSphereMQ.

4.3.2.2 A Simple Model for the Message Processing Time

Figure 4.9(a) shows that the message processing time depends only on the number of filters n_{fltr}. In contrast to the servers following the basic model and SunMQ, it does not depend on the replication grade r. Thus, the time to send messages is obviously so small that it is not noticeable for a replication grade of up to $r = 40$.



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



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

Figure 4.9: *WebSphereMQ: Impact of the number of filters n_{fltr} and the message replication grade r on the received message throughput – measurements and analytical data.*

Table 4.5: *WebSphereMQ: Empirical values for the model parameters of the message processing time in Equation (4.11).*

parameter	t_{rcv} (s)	t_{fltr} (s)	t_{tx} (s)
app. prop. filtering	$7.03 \cdot 10^{-4}$	$1.02 \cdot 10^{-5}$	0.0

A linear model like for the other servers does not work for the approximation of the measured results. Therefore, we propose the following model for the message processing time B:

$$B = t_{\text{rcv}} + n_{\text{fltr}} \cdot \sqrt{(n_{\text{fltr}})} \cdot t_{\text{fltr}} + r \cdot t_{\text{tx}} \ . \tag{4.11}$$

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

4.3.2.3 Validation of the Model by Measurement Data

As mentioned in Section 4.2.1, the received and the overall throughput can be analytically calculated by $\frac{1}{B}$ and $\frac{r+1}{B}$. Again, we adapt the model parameters t_{rcv}, t_{fltr}, and t_{tx} in Equation (4.11) by a least squares approximation and obtain them for the values in Table 4.5. As a characteristic for the WebSphereMQ, the result for the time to dispatch a message is a very small value. Therefore, we neglect it in our further evaluations and assume it as zero.

We calculated the received and overall throughput for all measured data points based on these values and Equation (4.11), and plot them with dashed lines in Figures 4.9(a) and 4.9(b). The throughput from our analytical model corresponds very well with our measurement data for all numbers of filters n_{fltr} and all replication grades r. The model predicts the overall message throughput of the server quite accurately for a wide range of realistic values for the parameters n_{fltr} and r.

As observed for the basic model, for higher replication grades than the one considered in the experiments, other parameter values might limit the server throughput performance. Thus, we can observe this behavior for the WebSphereMQ for replication grades larger than $r = 80$. To capture additional bottlenecks we propose an extension of our models in the next section.

4.4 Performance Model Considering Complex Filtering

Our main objective in this section is to characterize the impact of different filter types on the message throughput. 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. The ActiveMQ server is chosen due to its open source character. In case of faulty behavior, we are able to track the errors down, whether it is a measurement artefact or a software error.

Filters evaluate user defined message headers where we set searchable String-Properties 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 $n_{\text{fltr}}^{\text{pos}}$ subscribers filter for this value such that each message is replicated $r = n_{\text{fltr}}^{\text{pos}}$ times. The additional $n_{\text{fltr}}^{\text{neg}}$ filters do not match, but they cause additional workload on the server. Thus, altogether $m = n_{\text{fltr}}^{\text{pos}} + n_{\text{fltr}}^{\text{neg}}$ subscribers are connected to the server and they are distributed over two subscriber machines.

4.4.1 Complex Filter Design Options

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

Simple Filters

We already examined the impact of simple filters in Section 3.2.5 with the following experimental setup. The publishers send only messages with ID #0. As depicted in Figure 4.10, we install $n_{\text{fltr}}^{\text{pos}}$ matching filters searching for ID value #0. Additionally, we install $n_{\text{fltr}}^{\text{neg}}$ different non-matching filters that search for values between #1 and #$(n_{\text{fltr}}^{\text{neg}})$.

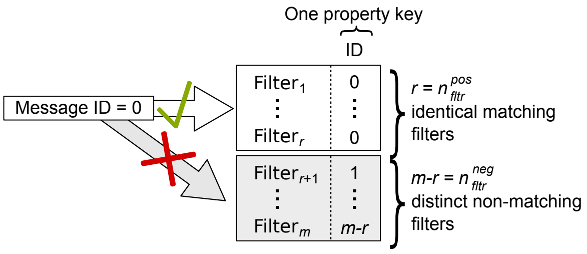


Figure 4.10: *Simple filters arrangement.*

Complex OR-Filters

We consider OR-filters with $n_{\text{cmp}}^{\text{fltr}}$ components. As illustrated in Figure 4.11, we install $n_{\text{fltr}}^{\text{pos}}$ identical 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 right, as defined by the JMS standard. The publishers send messages with ID #0 to produce a message replication grade of $r = n_{\text{fltr}}^{\text{pos}}$.

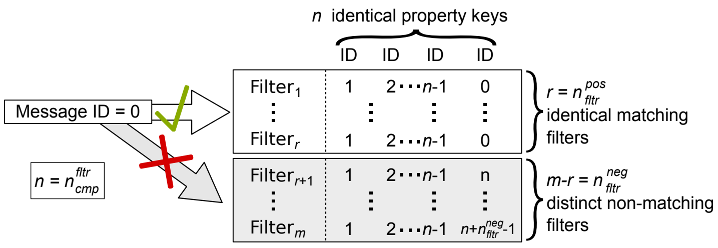

n identical property keys
ID ID ID ID
Message ID = 0
n = n^fltr_cmp
Filter1 1 2⋯n-1 0
Filterr 1 2⋯n-1 0
Filterr+1 1 2⋯n-1 n
Filterm 1 2⋯n-1 n+n^neg_fltr-1
r = n^pos_fltr identical matching filters
m-r = n^neg_fltr distinct non-matching filters

Figure 4.11: *Complex OR-filters arrangement.*

The last components of the $n_{\text{fltr}}^{\text{neg}}$ non-matching filters take values from $\#(n_{\text{cmp}}^{\text{fltr}})$ to $\#(n + n_{\text{fltr}}^{\text{neg}} - 1)$ with $n = n_{\text{cmp}}^{\text{fltr}}$.

Complex AND-Filters

We consider AND-filters with $n_{\text{fltr}}^{\text{len}}$ components. The publishers send messages with value #0 for each component ID$_i$, where $i = \{1; \ldots; n\}$ with $n = n_{\text{cmp}}^{\text{fltr}}$. As illustrated Figure 4.12, $n_{\text{fltr}}^{\text{pos}}$ subscribers install matching filters. The values set in the last component of the $n_{\text{fltr}}^{\text{neg}}$ non-matching filters take values between #1 and $\#n_{\text{fltr}}^{\text{neg}}$.

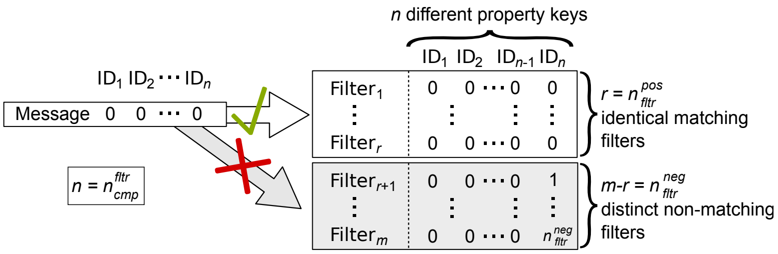


Figure 4.12: *Complex AND-filters arrangement.*

4.4.2 Results of the Measurement Experiments

We present the results for the experiments described in Section 4.4.1 for the parameters $n_{\text{fltr}}^{\text{pos}} \in \{1;2;5;10;20;40\}$, $n_{\text{fltr}}^{\text{neg}} = \{1;5;10;20;40;80;160\}$, and $n_{\text{cmp}}^{\text{fltr}} = \{1;2;4;8\}$.

The solid lines plotted in Figure 4.5 and in Figures 4.13–4.16 show the measured message throughput of the ActiveMQ JMS server. The received and the overall throughput is plotted in separate figure series. We observe in all experimental studies a similar behavior. With an increasing number of filters, the received and the overall throughput is only slightly reduced. 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 $50{,}000$ msgs/s. We take this observation into account for fitting the model parameters in the next section.

4.4.3 Extended Performance Model for the Message Processing Time

We use the measurement results from Section 4.4.1 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 $m = n_{\text{fltr}}^{\text{pos}} + n_{\text{fltr}}^{\text{neg}}$ and on their length $n_{\text{fltr}}^{\text{len}}$. 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

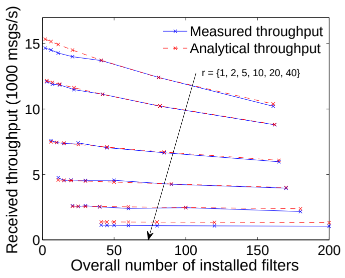
$$B = t_{\text{rcv}} + n_{\text{cmp}}^{\text{fltr}} \cdot m \cdot t_{\text{fltr}} + r \cdot t_{\text{tx}}. \tag{4.12}$$

The empirical service time can be derived from the received message throughput of the measurement results in Section 4.4.2. The parameters t_{rcv}, t_{fltr}, and t_{tx} are fitted to the proposed model by a least-squares approximation. We consider only those curves that are not limited by the $50{,}000$ msgs/s margin. The parameters are derived separately for the simple, complex OR-, and complex AND-filters. Table 4.6 summarizes their values. We observe that these empirical values of the model parameters are similar for all three experiment series.

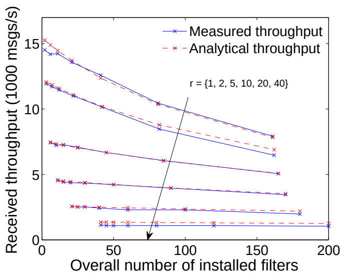
Table 4.6: *ActiveMQ: Empirical values for the parameters of the model given in Equation (4.12)*

	t_{rcv}	t_{fltr}	t_{tx}
Simple filters	$4.88 \cdot 10^{-5}$ s	$1.62 \cdot 10^{-7}$ s	$1.54 \cdot 10^{-5}$ 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

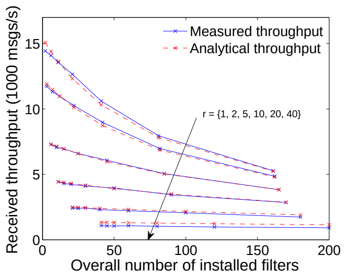
Based on the model and the parameters, we calculcate the analytical values for the received ($\frac{1}{B}$) and the overall throughput ($\frac{r+1}{B}$). They are plotted as dashed lines in Figure 4.5 and in Figures 4.13–4.16. For small values of the replication grade $r = \{1; 2; 5; 10\}$ the analytical data estimates the measured data very well. If the replication grade inreases, i.e., $r = \{20; 40\}$, the limit of $50{,}000$ msgs/s for the overall throughput of the server is reached and the analytical model tends to overestimate the measured throughput.



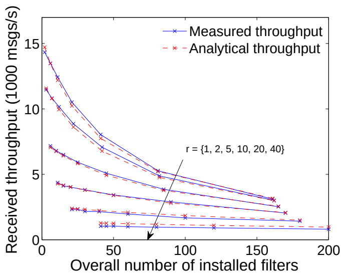
(a) Filter length $n_{cmp}^{fltr} = 1$.



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

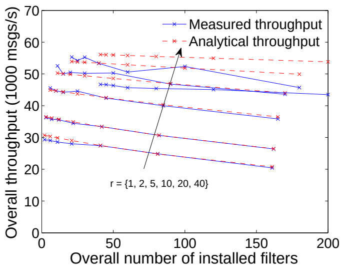


(c) Filter length $n_{cmp}^{fltr} = 4$.

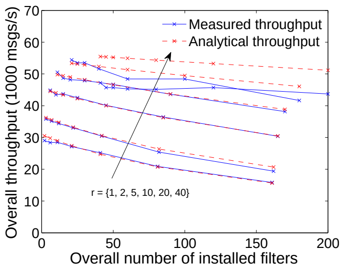


(d) Filter length $n_{cmp}^{fltr} = 8$.

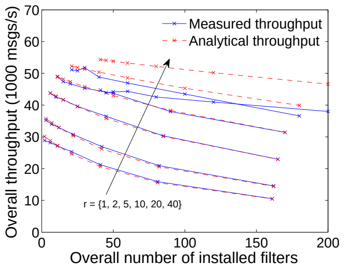
Figure 4.13: *ActiveMQ: Measured and analytical received message throughput for complex OR-filters depending on the message replication grade* r.



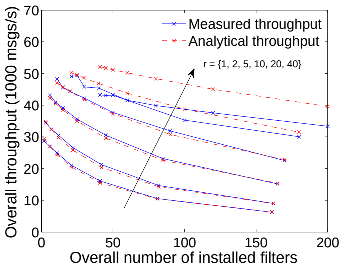
(a) Filter length $n_{cmp}^{fltr} = 1$.



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

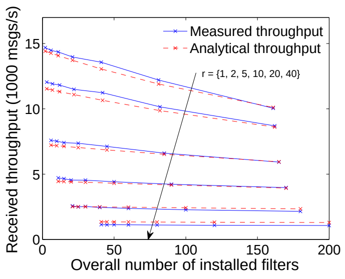


(c) Filter length $n_{cmp}^{fltr} = 4$.

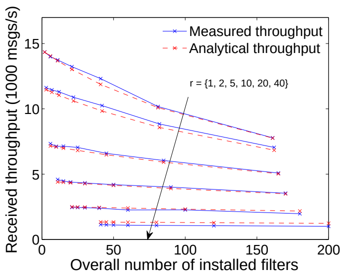


(d) Filter length $n_{cmp}^{fltr} = 8$.

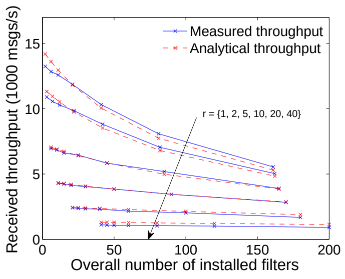
Figure 4.14: *ActiveMQ: Measured and analytical overall message throughput for complex OR-filters depending on the message replication grade* r.



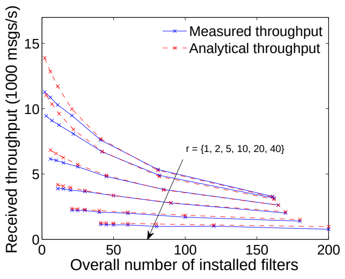
(a) Filter length $n_{cmp}^{fltr} = 1$.



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

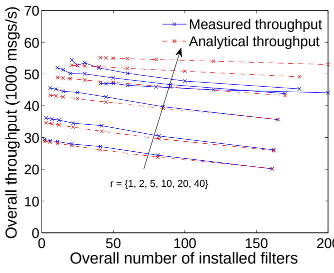


(c) Filter length $n_{cmp}^{fltr} = 4$.

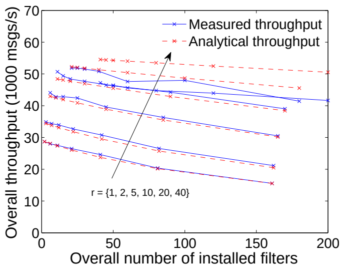


(d) Filter length $n_{cmp}^{fltr} = 8$.

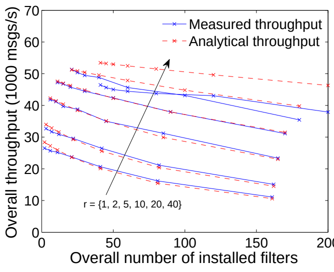
Figure 4.15: *ActiveMQ: Measured and analytical received message throughput for complex AND-filters depending on the message replication grade* r.



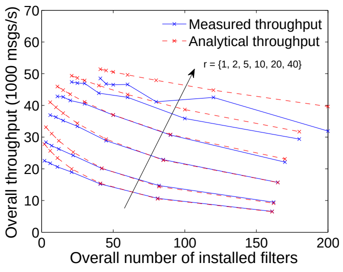
(a) Filter length $n_{cmp}^{fltr} = 1$.



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



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



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

Figure 4.16: *ActiveMQ: Measured and analytical overall message throughput for complex AND-filters depending on the message replication grade r.*

4.5 Application of the Models as Best-Practice Example

To conclude this chapter, the feasibility of the presented models is shown by an application of the evaluated models. This illustrates the benefits obtained from the analytical models for a system engineer. Overall, a summarizing evaluation and comparison of the performance for the different server types is given.

We assume a distributed notification service, i.e., producers generate so-called events and consumers are notified about them. A JMS server can be used to implement such a service. We assume many producers and 100 consumers. There are many event types, but each consumer is interested in only one. The consumers may use filters with $n_{\text{fltr}}^{\text{all}} = 100$ to get only the relevant events; otherwise, they are notified about all events and have to process a higher load. The consumers are interested in $n_{\text{fltr}}^{\text{diff}} \in \{1; 10; 100\}$ different events. We predict the JMS server throughput based on the results of our study, in particular for different message replication grades r. Large replication grades occur if several clients filter for the same events. If no filters are used, the throughput is determined by the results of the filter activation experiment as depicted in Figure 3.10. If filters are applied, we use Equation (4.9), Equation (4.10), and Equation (4.11) with the respective parameters for application property filtering to calculate the server capacity. We have compiled the throughput of received messages at the servers in Table 4.7.

The use of filters increases the received throughput performance in these application scenarios for FioranoMQ, ActiveMQ, Bea WebLogic, and for SunMQ, but not for WebSphereMQ. However, the use of filters is not only recommended to increase the server throughput but also to protect the consumers from undesired load if they are only interested in 1% or 10% of the messages. Considering the messages per second as a measure, we immediately realize that ActiveMQ and Bea WebLogic are superior to FioranoMQ and SunMQ in all considered application scenarios. WebSphereMQ is outperformed by all other server implementations. Since FioranoMQ, Bea WebLogic, and ActiveMQ are based on the same prediction model, we discuss only the performance of

Table 4.7: *Received throughput capacity of the FioranoMQ, SunMQ, WebSphereMQ, Bea WebLogic, and ActiveMQ JMS server for different application scenarios with 100 subscribers and an overall number of $n_{\text{fltr}}^{\text{all}} = 100$ filters if filters are used.*

$n_{\text{fltr}}^{\text{diff}}$	repl. grade r	Fior.MQ capacity (msgs/s)	SunMQ capacity (msgs/s)	WebSph. capacity (msgs/s)	Bea Web. capacity (msgs/s)	ActiveMQ capacity (msgs/s)
-	100	456	228	90	532	487
100	1	676	1817	85	6629	12416
10	1	676	2566	85	6629	12416
1	1	676	2676	85	6629	12416
10	10	615	1333	85	3260	4549

ActiveMQ and SunMQ. Without filters, ActiveMQ has about twice the capacity of SunMQ and each consumers receives all messages. With all consumers having a filter installed, the throughput increases to $12{,}416$ msgs/s for ActiveMQ, and for SunMQ to $1{,}817$, $2{,}566$, or $2{,}677$ msgs/s if the number of different filters $n_{\text{fltr}}^{\text{diff}}$ are 100, 10, or 1. This holds for a message replication grade of $r = 1$. In this case, the clients get only 1% of all messages. For a replication grade of $r = 10$, the clients get 10% of all messages. Then, ActiveMQ achieves a throughput of $4{,}549$ msgs/s and SunMQ $1{,}333$ msgs/s if $n_{\text{fltr}}^{\text{diff}} = 10$. Thus, ActiveMQ has four times the capacity of SunMQ if filters are applied.

Comparing the same performance measure for SunMQ and FioranoMQ, the performance of FioranoMQ is reduced by half, even if FioranoMQ is on the same performance level as ActiveMQ without any filters installed. These conclusions can also be drawn from the overhead values in Table 4.8, which are retrieved from the analytical models. FioranoMQ has the lowest overhead while receiving messages. ActiveMQ is very efficient in processing filters, followed by Bea WebLogic. Dispatching overhead is for all servers in the same order of magnitude, besides the WebSphereMQ, where the filter processing overhead dominates the overall performance.

After all, only ActiveMQ and Bea WebLogic can be considered as high

Table 4.8: *Comparison of the overhead times from the regression analysis.*

	Time to receive a message t_{rcv}	Time to process a filter t_{fltr}	Time to dispatch a message t_{tx}
WebSphereMQ	$7.0 \cdot 10^{-4}$s	$1.1 \cdot 10^{-5}$s	-
SunMQ	$1.1 \cdot 10^{-4}$s	$2.1 \cdot 10^{-6}$s	$4.0 \cdot 10^{-5}$s
FioranoMQ	$8.5 \cdot 10^{-7}$s	$7.0 \cdot 10^{-6}$s	$1.7 \cdot 10^{-5}$s
ActiveMQ	$4.9 \cdot 10^{-5}$s	$1.6 \cdot 10^{-7}$s	$1.5 \cdot 10^{-5}$s
Bea WebLogic	$8.0 \cdot 10^{-5}$s	$5.3 \cdot 10^{-7}$s	$1.7 \cdot 10^{-5}$s

throughput performance JMS platforms. From a throughput performance point of view, WebSphereMQ is clearly inferior to all others. However, WebSphereMQ comes with a variety of other functionalities. Thus, the mere consideration of the throughput performance of its JMS module is certainly not a sufficient criterion against this solution. In particular, if high throughput performance is not required, this kind of server might still be a good choice.

4.6 Concluding Remarks on Performance Models

In the previous sections, we presented some basic models for different JMS servers. An adaption of the basic model is given for servers differing from the standard behavior. We additionally extended the model to enhanced scenarios, like filter length or type.

Overall, we investigated the joint impact of the number of filters and the replication grade on the server capacity of FioranoMQ, ActiveMQ, Bea WebLogic, SunMQ, and WebSphereMQ. ActiveMQ, FioranoMQ, and Bea WebLogic lead to enhanced throughput for correlation ID filters compared to application property filters while the filter type does not lead to different results for SunMQ and WebshpereMQ. Only SunMQ implements an optimized filter matching algorithm such that identical filters can be handled more efficiently than different filters.

The message replication grade has an impact on the message processing time for all compared solutions, but not for WebSphereMQ as long as a replication grade of $r = 40$ is not exceeded. The filtering effort for ActiveMQ, Bea WebLogic, SunMQ, and FioranoMQ increases at most linearly with the number of installed filters, whereas WebshpereMQ shows a limited filter scalability in the experiments. As a consequence, our models for the message processing time have to be differentiated according to the server type. SunMQ and WebSphereMQ have a substantially different capacity model than the others.

In general, the models are useful to predict the server capacity for specific application scenarios. Thus, they can be used to dimension the number of servers in an application server network. The throughput comparison of the different server platforms helps to decide which of these solutions satisfies the requirements of a special distributed application from a performance point of view. The generic methodology used to evaluate the presented servers is not limited to them and can be easily applied for any other JMS or publish/subscribe server implementation. A major difference from classical benchmarking is the flexibility in choosing the set of evaluated parameters. This enables a focused analysis of important parameters, as in our scenarios the joint impact of the number of applied filters and the replication grade.

The evaluation of this aspect shows that there are several bottlenecks where the filtering, the heart of content based routing, is the crucial part of the system. We identified filtering in combination with the replication grade as one of the most important limiting factors for scalability.

Using the presented modeling methodology, a good estimate for the system throughput performance is achievable. However, another aspect for characterizing a JMS server is its interal system performance in terms of message queuing, delay, and configuration issues. These aspects are the focus of the following chapter.

5 Analytical Assessment of JMS Server Performance

In the previous chapters, we have measured the maximum message throughput of a JMS server depending on the number of clients, the number of installed filters, the filter type, and the replication grade of a message. We have learned from the prior experiments that both the number of filters and the replication grade impact the JMS server capacity in terms of message throughput. Based on our basic model for the mean $E[B]$ of the message processing time B and the mean $E[R]$ of the message replication grade R in a certain application scenario, we can predict a JMS server's capacity.

Our goal is to characterize the average waiting times of messages passing a JMS server. Therefore, we pick the FioranoMQ JMS server as an example and model it based on an $M/GI/1-\infty$ queuing system. Since both, additional filters and redundantly sent messages reduce the JMS server capacity, it is of interest to apply such a model in a realistic environment to monitor and predict performance bottlenecks in realtime. For calculating the waiting time *distribution function* (DF), the numerical inversion of its Laplace-Stieltjes transform is necessary which cannot be approximated with sufficient precision in reasonable time. Considering an on the fly evaluation scenario in order to predict and react on abnormal behavior, a fast response of such an algorithm is required. Even available approximation methods are numerically rather intractable, or they are specific to the used service time distribution. Therefore, we propose in a simple approximation of the waiting time DF for arbitrary service time DFs in Section 5.1. It is based on the Gamma-distribution, therefore, we denote it *Gamma-approximation*. It takes into account the first, second, and third moment of the service time DF.

In addition to the characterization of the JMS server throughput performance it is possible to improve system performance by avoiding load using an appropriate system architecture. So we compare two design alternatives for distributed JMS systems regarding their capacity, the so-called *publisher-side server replication* (PSR) and *subscriber-side server replication* (SSR).

We start this chapter with an abstract introduction and validation of the Gamma-approximation in Section 5.1. Then we review some basics about the $M/GI/1-\infty$ queuing system, and apply it to the FioranoMQ JMS server in Section 5.2. Using the queuing system, we analyze the message waiting time caused by the FioranoMQ server. Finally, we conclude this chapter with a comparison of the JMS system design alternatives PSR and SSR in Section 5.3.

5.1 Gamma-Approximation of the $M/GI/1-\infty$ Waiting Time

Many problems in telecommunication networks can be modeled by queuing systems. If the customers are flows, their inter-arrival time follows usually a Poisson process [44]. Thus, the analysis of the waiting time of the $M/GI/1-\infty$ queuing system is often required. The Takacs recursion formula allows a simple calculation of the k-th moments (cf. Equation (5.112) in [126]). The Pollaczek-Khintchine formula yields even the Laplace transform of the entire waiting time DF [140]. This formula can hardly be numerically evaluated although there are some methods [24, 28, 76]. Explicit expressions in the time domain exist only for a few special cases like the $M/M/1-\infty$, the $M/D/1-\infty$ queuing system [131], or some long-tail service time distributions [32]. Approximations of the waiting time DF exist for general service time DFs in $M/GI/1-\infty$ [145] and even for general inter-arrival time DFs, i.e., for $GI/GI/1-\infty$ [27]. However, they are quite complex and must be adapted for specific service time distributions, or provide feasible results only for specific parameter ranges.

Discrete time analysis (DTA) can be used to calculate the waiting time of any discrete time $GI/GI/1-\infty$ system with a finite inter-arrival and service time distribution [18]. We first show that DTA can be also used for the approximation of a continuous time $GI/GI/1-\infty$ queuing system. Then, we apply DTA to validate the accuracy of the waiting time DF obtained by the Gamma-approximation.

We apply the new approximation method for a wide range of coefficients of the variation $c_{\text{var}}[B]$ of the service time B and the system utilization ρ.

The remainder of this section is organized as follows. First, a review of existing approaches is given to calculate or approximate the waiting time distribution function of an $M/GI/1-\infty$ queuing system. Then, we propose the Gamma-approximation for this task.

5.1.1 Review of the $M/GI/1-\infty$ Queuing System

The $M/GI/1-\infty$ queuing system consists of a Poisson arrival process with rate λ, i.e., the inter-arrival time A of the customers is exponentially distributed (Markov, M) with mean $E[A]=\frac{1}{\lambda}$ and their service time B follows an *identically and independently distributed* (iid.) general distribution (GI) with mean $E[B]$. The k-th moments $E[W^k]$ of the waiting time W can be calculated by Takacs' recursion formula. However, the DF of the waiting time cannot be directly computed in the time domain. The Pollaczek-Khintchine solution provides a formula solely for its generating function [140]. The inverse transform into time domain is difficult and can be done analytically only for some special cases.

The Takacs Recursion Formula

The k-th moments of the waiting time of all customers can be calculated by the Takacs recursion formula in [126]:

$$E[W^k] = \frac{\lambda}{1-\rho} \cdot \sum_{i=1}^{k} \binom{k}{i} \cdot \frac{E[B^{i+1}]}{i+1} \cdot E[B^{k-i}], \tag{5.1}$$

with $\rho = \frac{E[B]}{E[A]}$ being the utilization of the system and $E[W^0] = 1$. Thus, the first and second moments of the waiting time are

$$E[W] = \frac{\lambda \cdot E[B^2]}{2 \cdot (1-\rho) \cdot E[B]}, \tag{5.2}$$

$$E[W^2] = 2 \cdot E[W]^2 + \frac{\lambda \cdot E[B^3]}{3 \cdot (1-\rho)}. \tag{5.3}$$

In particular, we need the first and second moment of the waiting time regarding only *waiting customers* (wc). As the waiting time probability is $p_{\text{w}} = \rho$, they are given by

$$E[W_{\text{wc}}] = \frac{E[W]}{p_{\text{w}}}, \tag{5.4}$$

$$E[W_{\text{wc}}^2] = \frac{E[W^2]}{p_{\text{w}}}. \tag{5.5}$$

The Pollaczek-Khintchine Solution

The *Laplace-Stieltjes transform* (LST) $X^*(s)$ of a DF $X(t)$ for a random variable X is defined by

$$X^*(s) = \int_0^\infty e^{-s \cdot t} dX(t). \tag{5.6}$$

The LST of the waiting time DF is given by the well-known formula by Pollaczek and Khintchine

$$W^*(s) = \frac{s \cdot (1-\rho)}{s - \lambda + \lambda \cdot B^*(s)} \tag{5.7}$$

with $B^*(s)$ being the LST of the service time DF. There are means to get numerical results from this expression [24, 76, 144], but the available tools are neither precise nor fast enough to calculate these numerical results, especially for the inverse of the LST.

Explicit DFs for Special Cases

For some special cases, it is possible to obtain the inverse transform of the formula in Equation (5.7) such that an explicit DF is available. We present two of these special cases in the following.

The waiting time DF for the $M/M/1-\infty$ system can be calculated by

$$W(t) = 1 - \rho \cdot e^{-(1-\rho) \cdot t/E[B]}. \tag{5.8}$$

The solution for $M/D/1-\infty$ is somewhat more complex and numerically challenging (cf. Equation (2.122) in [131]):

$$W(t) = 1 - (1-\rho) \cdot \sum_{n=m+1}^{\infty} e^{-\lambda \cdot (n \cdot E[B] - t)} \cdot \frac{\lambda^n}{n!} \cdot (n \cdot E[B] - t)^n \tag{5.9}$$

with $m = \lfloor \frac{t}{E[B]} \rfloor$. Other explicit solutions are given for a class of long-tail service time DFs in [32].

Approximative Solutions for the DF

There are also approximative solutions of the form

$$W(t) = 1 - (\alpha \cdot e^{-\beta \cdot t} + \gamma \cdot e^{-\delta \cdot t}) \tag{5.10}$$

if some preconditions regarding B are met (cf. Section 4.4.1 in [145]). The parameters α, β, γ, and δ are quite complex to calculate and there is not always a solution for them.

Another, simpler approximation for the waiting time DF of the general $GI/GI/1-\infty$ system is given in [27] of the form

$$W(t) = 1 - \alpha \cdot e^{-\eta \cdot t}. \tag{5.11}$$

The rate parameter η is approximated based on the properties of the service time DF and there are various specialized formulae to adapt η to the exact type of the service time. We can calculate the waiting time DF for $M/Gamma/1-\infty$ with the following parameters:

$$\begin{aligned} \eta &= \frac{2 \cdot (1-\rho)}{1 + c_{\text{var}}[B]^2} \cdot \left(1 - (1-\rho) \cdot \frac{1 - c_{\text{var}}[B]^2}{3 \cdot (1 + c_{\text{var}}[B]^2)}\right) && (5.12) \\ \alpha &= \eta \cdot E[W]. && (5.13) \end{aligned}$$

The latter equation assures the correct mean waiting time of the approximated DF. This approximation works well if ρ is sufficiently large. A similar approximation has been applied in [135] to calculate the quantiles of waiting times (cf. Section 1.3 in [135]).

In the following we present the Gamma-approximation. We validate the approximation by DTA. But DTA is limited to smaller values of ρ. Therefore, we use the approximation presented in this section for the validation of large values of ρ.

5.1.2 The Gamma-Approximation

We introduce first the Gamma-distribution and some of its properties. Then we use the first and second moment of the waiting time of the waiting customers in an $M/GI/1-\infty$ system to determine the α- and β-parameter of a Gamma-distribution to get an estimate for its DF in a tractable way.

The Gamma-Distribution

The base for the Gamma-*distribution* $\Gamma(\alpha, \beta)$ is the Gamma-*function* $\Gamma(z)$, which is defined by

$$\Gamma(z) \quad = \quad \begin{cases} 0 & \text{if } x < 0 \\ \int_0^\infty t^{z-1} \cdot e^{-t} dt & \text{if } 0 \leq t. \end{cases} \tag{5.14}$$

The most important properties of the Gamma-function $\Gamma(z)$ are

$$\Gamma(z+1) \quad = \quad z \cdot \Gamma(z) \text{ if } z > 0 \tag{5.15}$$

$$\Gamma(k+1) \quad = \quad k! \text{ if } k \in \mathbb{N}_0 \tag{5.16}$$

$$\Gamma\left(\frac{1}{2}\right) \quad = \quad \sqrt{\pi}. \tag{5.17}$$

These are interesting properties of the Gamma-function, but they are not required in the following. The Gamma-distribution $\Gamma(\alpha, \beta)$ is given by its *probability density function* (PDF)

$$f_{\Gamma(\alpha,\beta)}(t) \quad = \quad \begin{cases} 0 & \text{if } t < 0 \\ \frac{\beta^{-\alpha} \cdot t^{\alpha-1} \cdot e^{-t/\beta}}{\Gamma(\alpha)} & \text{if } t \geq 0. \end{cases} \tag{5.18}$$

The calculation of its DF $F_{\Gamma(\alpha,\beta)}$ and even of its inversion $F^{-1}_{\Gamma(\alpha,\beta)}$ is implemented by many tools for statistical analysis, e.g., in Matlab [142]. However, explicit solutions for $F_{\Gamma(\alpha,\beta)}$ exist only for integral values of α. Then, we have

$$F_{\Gamma(\alpha,\beta)}(t) \; = \begin{cases} 0 & \text{if } t < 0 \\ 1 - e^{-t/\beta} \cdot \sum_{0 \leq i < \alpha} \frac{(t/\beta)^i}{i!} & \text{if } t \geq 0. \end{cases} \tag{5.19}$$

Thus, the Gamma-distribution $\Gamma\left(k, \frac{1}{k \cdot \lambda}\right)$ equals the Erlang-distribution $Erlang(k, \lambda)$. Therefore, the Gamma-distribution can be viewed as an extension of the Erlang-distribution towards $k \in \mathbb{R}^+$. The mean, the variance, and the coefficient of variation of the Gamma-distribution are

$$\begin{aligned} E[X] &= \alpha \cdot \beta && (5.20) \\ \text{VAR}[X] &= \alpha \cdot \beta^2 && (5.21) \\ c_{\text{var}}[X] &= \frac{1}{\sqrt{\alpha}}. && (5.22) \end{aligned}$$

Hence, the Gamma-distribution may be used to approximate distributions with a given mean and variance.

Estimation of the $M/GI/1-\infty$ Waiting Time Distribution by the Gamma-Distribution

The first and the second moment of the waiting time of the waiting customers of an $M/GI/1-\infty$ queuing system can be calculated by Equations (5.4) and (5.5). We use them to set the parameters α and β of the Gamma-distribution after manipulating Equations (5.20) and (5.21) by

$$\begin{aligned} \alpha &= \frac{E[W]^2}{\text{VAR}[W]} = \frac{E[W]^2}{E[W^2] - E[W]^2}, && (5.23) \\ \beta &= \frac{E[W]}{\alpha}. && (5.24) \end{aligned}$$

The resulting DF $F_{\Gamma(\alpha,\beta)}$ describes then the distribution of the waiting time of customers that are not immediately served upon arrival. The waiting time distribution of all customers is thus given by

$$F(t) = 1 - \rho + \rho \cdot F_{\Gamma(\alpha,\beta)}(t). \qquad (5.25)$$

If the service time in $M/GI/1–\infty$ is exponential, we get the simple $M/M/1–\infty$ system. As the waiting time DF of its waiting customers is exponential, we have $c_{\text{var}}[W] = 1$, and therefore, $\alpha = 1$ (cf. Equation (5.22)). In this case, the Gamma-approximation meets the exponential distribution exactly with the same mean $E[W]$. The question arises: how exact is the Gamma-approximation for other distributions of the service time? This is the issue we discuss in the validation part. To that end, we compare its results with the ones of the DTA and the approximation given in Equation (5.13).

5.1.3 Discrete Time Analysis and its Accuracy

In this section, we explain the *discrete time analysis* (DTA) for the discrete time $GI/GI/1$–D^{max} queuing system with bounded delay D^{max} [18,26,27]. We use it to approximate the continuous time $GI/GI/1$–∞ queue and identify potential sources of inaccuracies. Finally, we compare its results for the waiting time DF of an $M/D/1-\infty$ and an $M/M/1-\infty$ queuing system with analytical results and show that its accuracy depends on the parameters of the DTA-approximation.

DTA of the $GI/GI/1-D^{max}$ Queuing System with Bounded Delay D^{max}

The discrete time $GI/GI/1-D^{max}$ queuing system with bounded delay D^{max} is based on discrete time units, i.e., the inter-arrival time of its customers is distributed according to an iid. general distribution. Also their holding time follows an iid. general distribution. The respective random variables are denoted by A and B. The value range of both distributions contains only multiples of a common basic time unit u.

State Transitions of the $GI/GI/1$–D^{max} Queue: We analyze the discrete time $GI/GI/1-D^{max}$ queue by considering a *discrete time Markov chain* (DTMC) whose state represents the unfinished work in the buffer which is described by the random variable U. Upon arrival of a new customer, the unfinished work in the buffer is incremented by the new customer's service time B. If this exceeds the delay bound D^{max} of the buffer, the unfinished work is set to this delay bound. Afterwards, the unfinished work is decreased by the passing time units until the next customer arrives. Renewal points of the process exist shortly before (-) and after (+) the arrival instants. We number them t_n^- and t_n^+ and the states U_n^- and U_n^+, accordingly. The Markov chain evolves based on the following recursive stochastic equations.

$$\begin{aligned} U_{n+1}^- &= \max(U_n - A, 0) && (5.26) \\ U_{n+1}^+ &= \min(U_{n+1} + B, D^{\max}). && (5.27) \end{aligned}$$

Discrete Time Analysis: An early use of *discrete time analysis* (DTA) can be found in [18, 19, 21, 23, 26] with application to packet networks. The concept of DTA works as follows. An iteration algorithm starts with a distribution x_0 of the system state at the first renewal point. The distribution x_{n+1} of the system state at renewal point $n+1$ is calculated based on the distribution x_n of the system state at renewal point n and the distribution y of the factors. The calculation itself is described by a state transition function from one renewal point to the next one, i.e., it is denoted by the recursive stochastic equation

$$X_{n+1} = f(X_n, Y). \tag{5.28}$$

If the Markov chain is aperiodic, the series of the x_n converges to the stationary state distribution which characterizes the distribution of the system states at the renewal points after a long time. We recognize convergence in practice if the entries of two successive state distributions x_n and x_{n+1} differ not more than ε_c, where typically $\varepsilon_c < 10^{-6}$. If the Markov chain is periodic, there are modifications to the iteration algorithm such that the series x_n converges also to the stationary distribution [73]. The whole concept is extended to different types of renewal points, e.g., shortly before and after a customer arrival, and stationary distributions can be calculated for both types. Thus, we can calculate the distribution of the unfinished work in the buffer shortly before and after a customer arrival by DTA using Equations (5.26) and (5.27) as state transition functions. Note that the stationary state distribution shortly before a customer arrival yields the waiting time distribution for new customers.

Approximation of the Continuous Time $GI/GI/1-\infty$ by Discrete Time $GI/GI/1-D^{\text{max}}$ through DTA

The continuous time $GI/GI/1-\infty$ queue and the discrete time $GI/GI/1-D^{\text{max}}$ queue with bounded delay D^{max} differ significantly regarding the nature of their inter-arrival time and service time distribution and regarding their buffer size. In addition, DTA is a numerical algorithm that terminates at a previously given threshold. This might lead to inaccurate results. As these issues may lead to wrong approximation results, actions must be taken to keep the error small. Table 5.1 shows an overview on the adjustable parameters for the DTA. Smaller values for all types ε increase the accuracy but also increase the computational costs as a consequence.

Continuous and Discrete Time Distributions: The DTA-approximation requires the transform of the continuous DF F_c of the inter-arrival time and the service time into discrete DFs F_d which are step functions. We achieve this by increasing the step function F_d to F_c at the multiples of the basic time unit u. Thus, the approximation quality can be increased by decreasing the basic time unit u.

Infinite and Finite Distributions: The range of the inter-arrival time and the service time may be infinite for the continuous time $GI/GI/1-\infty$ queue, but it must be limited for the discrete time $GI/GI/1-D^{\text{max}}$ queue since this is a requirement of the DTA algorithm. Thus we choose a value for t_{max}, such that $1-F_c(t_{\text{max}}) < \varepsilon_f$ holds and set F_d to

$$F_d(t) = \begin{cases} 0 & \text{for } t = 0 \\ F_c(n \cdot u) & \text{for } t < t_{\text{max}} \wedge t \in ((n-1) \cdot u, n \cdot u] \\ 1 & \text{for } t \geq t_{\text{max}}. \end{cases} \tag{5.29}$$

Obviously, the approximation quality can be increased by decreasing ε_f.

Infinite Buffer and Limited Delay: The continuous time $GI/GI/1-\infty$ queue has an infinite buffer by definition, but the discrete time $GI/GI/1-D^{\text{max}}$ queue requires a limited delay which lies in the nature of DTA. This is obviously a source for approximation errors. To avoid them, the delay bound D^{max} must be set large enough, i.e., that the probability ε_d for a customer to exceed this delay bound D^{max} is very small. This can be controlled by having a look at the stationary state distribution of the unfinished work shortly after the packet arrival. If the probability for the unfinished work to be D^{max} is smaller than ε_d, then the delay bound is sufficiently large. Thus, the approximation quality can be increased by decreasing ε_d.

Table 5.1: *Parameters of the DTA and their impact.*

Parameter	Description
ε_c	Stop criterion for the DTA.
ε_d	Maximum probability for loss rate.
ε_f	Quantile up to which a distribution is descretized.

Convergence Accuracy: The iteration algorithm of the DTA terminates if the difference of two consecutive distributions regarding the same renewal point differ less than ε_c in each component. Thus, the approximation quality can be increased by decreasing ε_c.

Validation of the DTA-Approximation of $GI/GI/1-\infty$

We first explain the generation of the factor distributions for the DTA and then compare the resulting DFs. Then, we show that the DTA-approximation of the discrete time $GI/GI/1-D^{\text{max}}$ system can be used to calculate the waiting time DF of the continuous time $GI/GI/1-\infty$ system. To that end, we validate its results by analytical values in the special cases of the $M/D/1-\infty$ and the $M/M/1-\infty$ system.

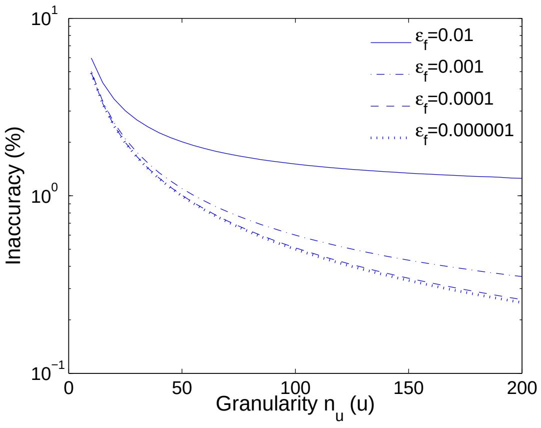
Generation of the Factor Distributions for DTA: We determine the time granularity by defining $E[A] = n_u \cdot u$ and choose $n_u = 100$ by default. As the DTA requires distributions instead of DFs to describe the factors A and B, we calculate them by

$$P(A' = k) = \begin{cases} 0 & \text{for } k = 0 \\ F_c^A(k) - F_c^A(k-1) & \text{for } k > 0. \end{cases} \tag{5.30}$$

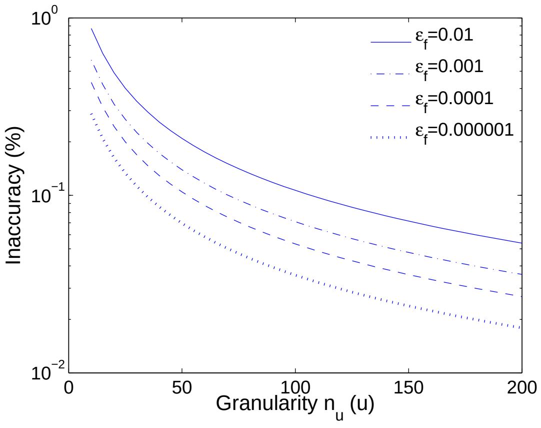
Note that the value k corresponds to a duration of $k \cdot u$. We limit the DFs for A and B according to Equation (5.29) with $\varepsilon_f = 10^{-4}$ by default and mark the discretized random variables by a single quote, e.g., A' is the discretized value of A. We test the system under a given utilization ρ. Thus, the mean of the service time is $E[B] = \rho \cdot E[A]$.

Due to the discretization, the mean $E[X]$ and the coefficient of variation $c_{\text{var}}[X]$ of the discretized A' and B' differ slightly from the ones of A and B. The discretization error of A' and B' leads also to a slightly different system utilization $\rho' = \frac{E[B']}{E[A']}$. In general, there are different discretization methods possible to keep the error small. The precision of the methods depends on the discretization unit and the shape of the DF, as discussed in [20].

We use the exponential DF $F_c(t) = 1 – e^{-t/E[A]}$ as the base for the inter-arrival time DF for A'. Figure 5.1 illustrates the inaccuracy of the discretized exponential distribution A' in relation to the continuous DF of A. Figure 5.1(a) shows the discretization error $R_E[A'] = \frac{E[A'] - E[A]}{E[A]}$ regarding the mean $E[A]$. The discretized distribution has a slightly larger mean, but the difference decreases with increasing n_u. Decreasing the discretization parameter ε_f can improve the result only up to $\varepsilon_f = 0.0001$. Figure 5.1(b) shows the discretization error $R_{c_{\text{var}}}[A'] = \frac{c_{\text{var}}[A'] - c_{\text{var}}[A]}{c_{\text{var}}[A]}$ regarding the coefficient of variation $c_{\text{var}}[A]$. The discretized distribution has a slightly larger coefficient of variation, but the difference decreases with increasing n_u. In contrast to the mean, the accuracy of the coefficient of variation is further decreased by a decreasing ε_f.



(a) Relative error of the mean rate $R_E[A']$.

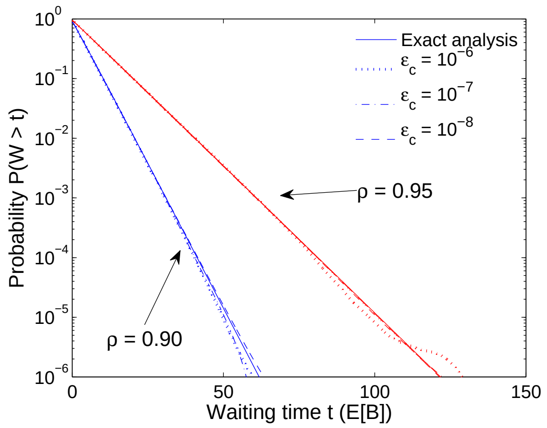


(b) Relative error of the coefficient of variation $R_{c_{\text{var}}}[A']$.

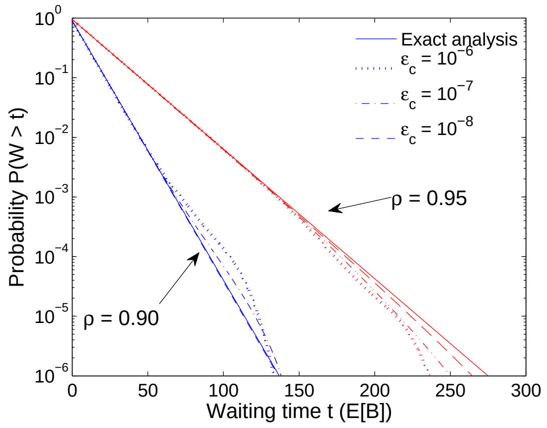
Figure 5.1: *Discretization error for an exponential DF depending on the discretization parameters n_u and ε_f.*

Comparison of Analytical and Approximated Distribution Functions: We compare the waiting time DFs for an $M/D/1-\infty$ and an $M/M/1-\infty$ system calculated from the DTA-approximation of the $GI/GI/1-D^{max}$ system and from the analytical formulae given in Equations (5.8) and (5.9). They are presented for a system utilization of $\rho = 0.90$ and $\rho = 0.95$ in Figure 5.2. We have plotted the *complementary cumulative DF* (CCDF) since this makes the difference between the analytical and approximated values more visible on a logarithmic y-scale. The waiting time is given in multiples of the mean service time $E[B]$ since this is the invariant component in most systems. The approximations are shown for $n_u = 100$, $\varepsilon_f = 10^{-4}$, $\varepsilon_d = 10^{-10}$, and $\varepsilon_c = 10^{-\{6,7,8\}}$ for both considered queuing systems. The approximation with $\varepsilon_c = 10^{-6}$ yields significant deviations for large waiting times. However, $\varepsilon_c = 10^{-7}$ yields already a sufficiently good correspondence between the analytical and approximative CCDF and the curve for $\varepsilon_c = 10^{-8}$ coincides with the analytical values. We observe within each of the figures that the inaccuracy increases for the same ε_c with increasing system utilization. When we compare the results for $M/D/1-\infty$ and $M/M/1-\infty$, we also realize that the inaccuracy increases for increasing coefficients of variation $c_{\text{var}}[B]$ of the service time, too. Thus, the DTA leads to good and trustworthy results only for small or medium coefficients of variation $c_{\text{var}}[B]$ and moderate system utilization.

From these observations, we use the discretization and termination parameters set to $n_u = 100$, $\varepsilon_f = 10^{-4}$, $\varepsilon_d = 10^{-10}$, and $\varepsilon_c = 10^{-8}$ for the DTA analysis in the following.



(a) $M/D/1-\infty$ system.



(b) $M/M/1-\infty$ system.

Figure 5.2: *Analytical and approximated CCDFs of the waiting time.*

5.1.4 Validation of the Gamma-Approximation

In this section, we illustrate the accuracy of the Gamma-approximation by comparing its complementary cumulative DF (CCDF) with the results obtained from the corresponding DTA-analysis in Section 5.1.3 and approximative results from the simple exponential $GI/GI/1-\infty$ approximation in Equation (5.13). We consider first systems with a different coefficient of variation $c_{\text{var}}[B]$ of the service time and different utilization levels ρ. Then, we study the impact of their third moment on the approximation accuracy for which we use a symmetric and a strongly asymmetric service time distribution.

Impact of the First and Second Moment of the Service Time

The first moment of the service time determines the system utilization $\rho = \frac{E[B]}{E[A]}$ and the second moment determines the coefficient of variation of the service time by $c_{\text{var}}[B] = \frac{\sqrt{E[B^2]-E[B]^2}}{E[B]}$. Since ρ and $c_{\text{var}}[B]$ are more intuitive, we use them to control our parameter studies instead of $E[B]$ and $E[B^2]$. We use the Gamma-distribution as service time since it can be easily adapted to meet a given ρ and $c_{\text{var}}[B]$ (cf. Equations (5.23) and (5.24)). We discretize the continuous Gamma-DF according to Equation (5.29) to obtain approximated and finite DF A' and B' as input for the DTA-analysis. For the sake of a fair comparison, we use $E[B']$, $E[(B')^2]$, and $E[(B')^3]$ to calculate the first and the second moment of the waiting time of waiting customers in Equations (5.4) and (5.5) since they are required to fit the parameters α and β of the Gamma-distribution in Equations (5.23) and (5.24). Then, we use this distribution together with $\rho' = \frac{E[B']}{E[A']}$ to derive the Gamma-approximation in Equation (5.25).

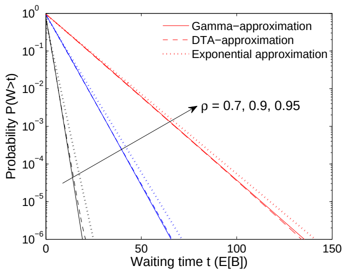
The four different parts of Figure 5.3 compare the CCDFs of the waiting time for the Gamma-approximation, the DTA-approximation, and the exponential approximation of Equation (5.13). We have chosen the coefficients of variation of the service time $c_{\text{var}} = \{0.05; 1.5; 2.0; 4.0\}$ in these figures to perform a parameter study. Note that the Gamma-approximation cannot be parameterized to approximate the waiting time DF when the coefficient of the service time is

$c_{\text{var}}[B]=0$. For $c_{\text{var}}[B]=1$ it yields exactly the analytical results. Therefore, we omitted this figure. The different values for the system utilization $\rho = \frac{E[B]}{E[A]} = \{0.7; 0.9; 0.95\}$ within a single figure are obtained by varying the mean service time $E[B]$.

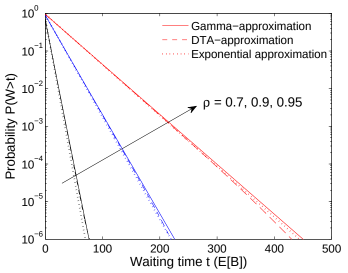
The DTA-approximation is very good for low values of the system utilization ρ and the solid lines of the Gamma-approximation coincides with the dashed lines of the DTA-approximation in all figures for $\rho = 0.7$ whereas the exponential approximation shows significant deviations. It is known from [27] that the accuracy of the exponential approximation is only good for large values of ρ. Thus, the Gamma-approximation yields very good approximation results for small and medium system utilization. The DTA-approximation is quite accurate for low coefficients of variation like $0 \leq c_{\text{var}}[B] \leq 1$ and the solid lines of the Gamma-approximation coincide with the dashed lines of the DTA-approximation for $c_{\text{var}}[B]=0.05$. For larger values of $c_{\text{var}}[B]$ and large values of ρ, the accuracy $\varepsilon_c = 10^{-8}$ of the DTA-approximation does not suffice to produce accurate results since the respective CCDFs deviate significantly from a straight line in the logarithmic plot. In these cases, the correspondence of the solid lines for the Gamma-approximation and the dotted lines of the exponential approximation is relatively good. Hence, the Gamma-approximation leads to good approximation result for a very broad range of $c_{\text{var}}[B]$ and ρ. The high resolution of our numerical results proposes that it can be used to calculate even large quantiles of the waiting time, e.g. the 99.999% percentile.

Impact of the Third Moment of the Service Time

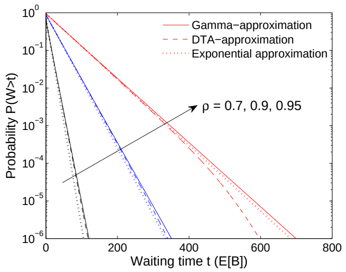
The Takacs formula requires the third moment of the service time to calculate the second moment of the waiting time of waiting customers (cf. Equation (5.5)). Therefore, we are interested in the impact of this value on the CCDF of the waiting time of an $M/GI/1-\infty$ and in the approximation accuracy of the Gamma-approximation regarding this parameter.



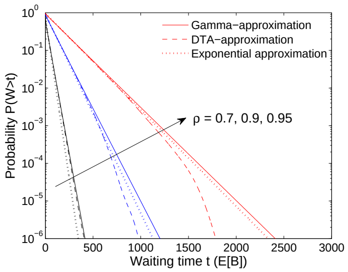
(a) $c_{\text{var}}[B] = 0.05$.



(b) $c_{\text{var}}[B] = 1.5$.



(c) $c_{\text{var}}[B] = 2.0$.



(d) $c_{\text{var}}[B] = 4.0$.

Figure 5.3: *The CCDFs of the waiting time from the Gamma-, DTA-, and exponential approximation for an $M/Gamma/1-\infty$ system with various coefficients of variation $c_{\text{var}}[B]$ at various system utilization levels ρ.*

Table 5.2: *Symmetric and asymmetric distributions, both with a first and second moment of $E[B]=100$ u and $E[B^2]=18000$ u^2.*

$B_{sym}=i$ (u)	$P(B_{sym}=i)$	$B_{asym}=i$ (u)	$P(B_{sym}=i)$
10	$\frac{40}{81}$	90	$\frac{80}{81}$
100	$\frac{1}{81}$		
190	$\frac{40}{81}$	900	$\frac{1}{81}$

To that end, we consider two distributions with the same first and second moment. They are given in Table 5.2. The symmetric distribution has a third moment of $E[B^3_{sym}]=3.6\cdot 10^6$ u^3 while the asymmetric distribution has a third moment of $E[B^3_{asym}]=9.72\cdot 10^6$ u^3. We use both service time distributions to calculate the CCDF of the waiting time for a system utilization of $\rho=0.9$, i.e., we set the mean of the inter-arrival to $E[A]=\frac{E[B]}{\rho}$. Figure 5.4 shows that the CCDFs of the waiting times from the service times B_{sym} and B_{asym} differ notably, but it also shows that both the DTA and the Gamma-approximation account for this difference.

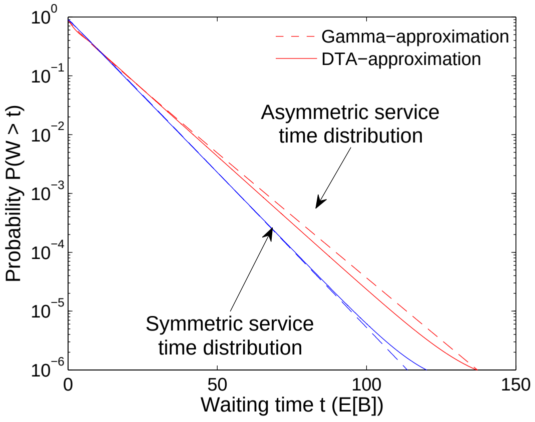


Figure 5.4: *The CCDFs of the waiting time from the Gamma- and DTA-approximation for an $M/GI/1-\infty$ system for $\rho=0.9$.*

5.1.5 Concluding Remarks on Gamma-Approximation

The review of the general $M/GI/1$–∞ queueing system and of some approaches to characterize the waiting time of waiting customers shows that there is no explicit expression except for some special cases. The waiting time DF can only be obtained by a numerical inversion of its Laplace-Stieltjes transform. Approximation methods are numerically also not simple, or they are specific to the used service time distribution. The proposed Gamma-approximation estimates the waiting time DF by a Gamma-distribution based on the first three moments of the service time distribution. The computation time for the Gamma-approximation is negligible while the DTA-analysis takes minutes or hours to produce sufficiently accurate results, and for large coefficients of variation $c_{\text{var}}[B]$ and a large utilization ρ it takes even days. This makes the advantage of the new calculation method obvious: it provides quite accurate estimates very quickly. Therefore, it is suitable for the implementation in real-time systems. We can use it to provide a prediction model for the waiting times of messages in a JMS server. But it can also be used in real-time systems where QoS measures like quantiles of the waiting time are needed to perform admission control. A good accuracy of the new Gamma-approximation has been shown, by the study of a broad range of coefficients of variation $c_{\text{var}}[B]$ of the service time and the system utilization ρ. The third moment of the service time has a minor impact on the CCDF of the waiting time, but it is also well covered by the Gamma-approximation.

After all, the Gamma-approximation is very simple to apply if the Gamma-distribution is available, which is the case in most of today's mathematical toolboxes. The fast calculation speed makes it appropriate for application in technical control systems, like a JMS server supporting high message throughput rates. In the next section, we will introduce a model for the message waiting time of a JMS system based on an $M/GI/1-\infty$ system and apply the Gamma-approximation to calculate the numerical results.

5.2 Analytical Performance Evaluation of the JMS Server Capacity

The models presented in Chapter 4 consider only average values regarding the message throughput rates. To characterize the server performance in a broader way, it is necessary to evaluate the time dynamic behavior, which can be described by internal queuing or even blocking behavior. Based on the performance model and parameters obtained for the FioranoMQ JMS server in Section 4.2.2.1, we investigate the server by a message waiting time analysis and by careful queuing theoretical observations.

5.2.1 JMS Server Capacity

First, we investigate the mean message processing time depending on the number of filters, to get a first hint concerning for the capacity of the FioranoMQ JMS server. Second, we introduce an approach to predict the server's capacity.

Average Message Service Time

With the basic performance model for the message service time in Equation (4.9) it is clear that the message service time increases linearly with the number of filters. Figure 5.5 illustrates the mean message service time $E[B]$ depending on the number of filters n_{fltr} and the average replication grade $E[R]$. The results are shown for correlation ID filtering and application property filtering. For small values of n_{fltr}, the mean message service time $E[B]$ is dominated by the average replication grade $E[R]$ but for large values of n_{fltr} the linear growth clearly dominates the influence of the message replication grade. Note that both the x- and the y-axis have a logarithmic scale. Thus, the service time for a message ranges over several orders of magnitude, which is due to different message replication grades, due to the linear growth of $E[B]$ with n_{fltr}, and due to filter type specific values of t_{rcv}, t_{fltr}, and t_{tx}. Hence, it is strongly application scenario specific.

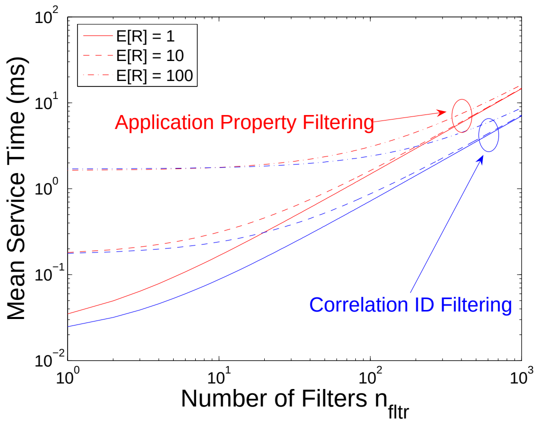


Figure 5.5: *Impact of the number of filters n_{fltr}, the average replication grade $E[R]$, and the filter type on the average message service time $E[B]$.*

Server Capacity

We define the server capacity by the maximum supportable load in terms of messages per second. If we allow a server CPU utilization of ρ, we can compute the server capacity in terms of received message throughput by

$$\lambda^{\max} = \frac{\rho}{E[B]}. \tag{5.31}$$

Figure 5.6 shows the server capacity for a maximum server CPU utilization of 90% for the same application scenarios like above. It shows the server capacity $\lambda^{\max}$ depending on the same parameters like above but for the sake of clarity we omitted the results for correlation ID filtering. Similar to the service time, the server capacity ranges over several orders of magnitude. It is obvious that the server capacity decreases both with an increasing number of filters n_{fltr} and with an increasing average replication grade $E[R]$. Filters protect the subscribers

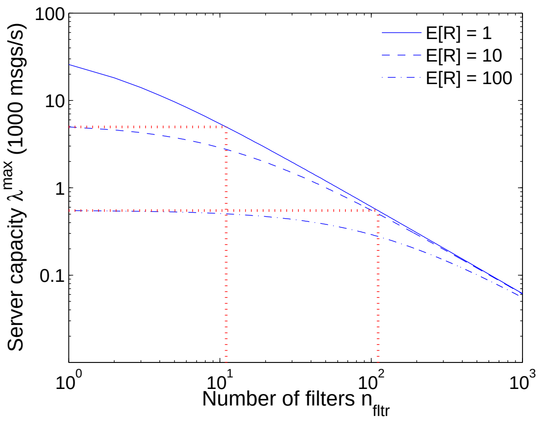


Figure 5.6: *Impact of the number of filters n_{fltr} and the average replication grade $E[R]$ on the server capacity λ^{max} for a maximum server CPU utilization of $\rho = 90\%$.*

from undesired messages, they reduce the replication grade, which limits the network traffic and improves the server capacity. However, the latter objective is not always achieved, which is also shown in Figure 5.6. Since the x-axis is scaled logarithmically, the figure starts with one filter applied. But if we assume a message replication grade of $E[R] = 10$ (100) without filters, it leads to the same capacity reduction like a message replication grade of $E[R] = 1$ and $n_{\text{fltr}} = 22$ (240) filters.

This leads to the question: when should a filter be applied to maximize the server throughput? We consider an information consumer q that has installed n_{fltr}^q filters on the server. Furthermore, we assume that these filters receive the proportion p_{match}^q of all messages. On the one hand, the filters increase the message processing time by $n_{\text{fltr}}^q \cdot t_{\text{fltr}}$ but on the other hand, they reduce it by

$(1-p^q_{\text{match}}) \cdot t_{\text{tx}}$. Thus, these filters increase the server capacity if the following inequality holds:

$$n^q_{\text{fltr}} \cdot t_{\text{fltr}} \quad < \quad (1-p^q_{\text{match}}) \cdot t_{\text{tx}}. \tag{5.32}$$

Taking the values of Table 4.1 into account, a single or two correlation ID filters ($n^q_{\text{fltr}} \in \{1;2\}$) should be used if their match probability is smaller than 58.7% or 17.4%, respectively. Three or more filters per consumer slow down the server more than forwarding any message if no filters are set. A single application property filter ($n^q_{\text{fltr}} = 1$) should be used if its match probability is smaller than 9.9%. Like above, two or more filters per consumer cannot lead to a capacity increase of the JMS server. However, filters are primarily used to protect the consumers against too many unwanted messages and the network against overload.

5.2.2 Modeling of the Message Waiting Time

The objective of this section is the investigation of the message waiting time. We first model the JMS server by a simple queuing system and discuss various distribution models for the message replication grade which impacts the variability of the service time. Then, we study the mean, the distribution, and in particular the 99% and the 99.99% quantiles of the message waiting time depending on the average server utilization.

A Simple Queuing Model for JMS Servers

For all servers we tested, the major part of the messages are queued at the publisher site due to a kind of push-back mechanism. As a consequence, we did not detect any message loss due to buffer overflow at the JMS server. In the following we consider only the the FioranoMQ server. In our experiments, we use permanently sending publishers that are only slowed down by the push-back mechanism of the JMS server. However, in reality, the arrival process is

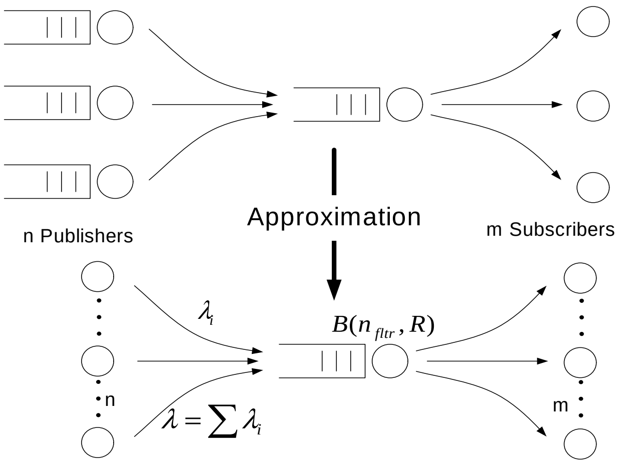


Figure 5.7: *A simple queueing model for a JMS server: $M/GI/1{-}\infty$.*

stochastic, i.e., the publishers do not send in a saturated manner. If the JMS server is not overloaded and if its message buffer is large enough to absorb all arriving messages, we can well approximate the complex overall system by a single message queue at the JMS server site. This is depicted by Figure 5.7. Furthermore, we assume a Poisson model for the arrival process in the busy hour, i.e., the inter-arrival times are exponentially distributed and the message arrival rate is denoted by λ. The arrival rate $\lambda = \sum_{0 \leq i < n} \lambda_i$ for that queue is the sum of the message rates λ_i from all publishers. This is a reasonable assumption since technical processes are often triggered by human beings. We can consider busy hour scenario of a system and assume that the arrival rate λ is constant.

Messages are served sequentially by the server with their processing time B. This random variable follows a general distribution. Thus, we can model the system by an $M/GI/1-\infty$ queue. The first $E[W]$ and second moment $E[W^2]$ of the message waiting time in this queuing system are given by Equation (5.2) and Equation (5.3), respectively. The utilization of the server is defined by $\rho = \lambda \cdot E[B]$ [140].

Model for the Message Service Time

The equations for the first two moments of the message waiting time require the first three moments of the message service time. The service time B for a message is composed of a constant part $D = t_{\text{rcv}} + n_{\text{fltr}} \cdot t_{\text{fltr}}$ and a variable part $V = R \cdot t_{\text{tx}}$ such that the first three moments can be calculated by:

$$\begin{aligned}
E[B] &= E[D+V] = D + E[R] \cdot t_{\text{tx}} \,, && (5.33)\\
E[B^2] &= E[(D+V)^2] = D^2 + D \cdot t_{\text{tx}} \cdot E[R] \\
&\quad + t_{\text{tx}}^2 \cdot E[R^2] \,, && (5.34)\\
E[B^3] &= E[(D+V)^3] = D^3 + 3 \cdot D^2 \cdot t_{\text{tx}} \cdot E[R] \\
&\quad + 3 \cdot D \cdot t_{\text{tx}}^2 \cdot E[R^2] + t_{\text{tx}}^3 \cdot E[R^3] \,. && (5.35)
\end{aligned}$$

To conduct a parameter study of the waiting time distribution depending on the mean $E[B]$ of the service time B and its coefficient of variation

$$c_{\text{var}}[B] = \frac{\sqrt{E[B^2] - E[B]^2}}{E[B]} \,, \tag{5.36}$$

we calculate the required $E[R]$ from Equation (5.33), and use $E[R]$ and Equation (5.34) to calculate $E[R^2]$. Depending on the appropriate model for the message replication grade R, we get $E[B^3]$ by using Equation (5.35) and the third moment of the respective distribution for the replication grade. In the following, we discuss various distributions to model the replication grade R.

Deterministic Distribution: If the replication grade r is constant, the distribution of the message processing time B is also deterministic and its coefficient of variation is $c_{\text{var}}[B] = 0$. Furthermore, the second and third moments of the message replication grade are

$$E[R^2] = E[R]^2 , \tag{5.37}$$

$$E[R^3] = E[R]^3 . \tag{5.38}$$

This model is very static and probably not appropriate to characterize real world scenarios.

Scaled Bernoulli Distribution: With a probability of p_{match}, a message is forwarded by all n_{fltr} filters and with a probability of $1 - p_{\text{match}}$, the message is not forwarded at all. This can be modeled by a scaled Bernoulli distribution. The corresponding first two moments are

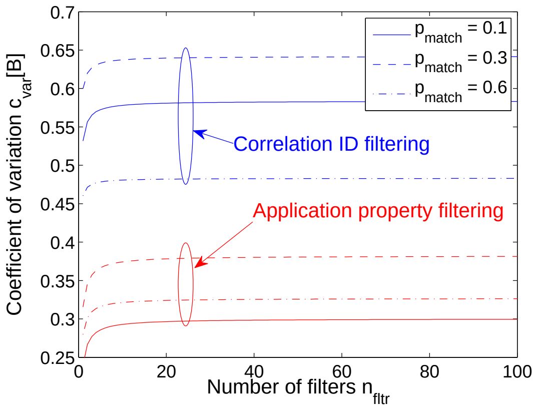
$$E[R] = p_{\text{match}} \cdot n_{\text{fltr}} , \tag{5.39}$$

$$E[R^2] = p_{\text{match}} \cdot n_{\text{fltr}}^2 . \tag{5.40}$$

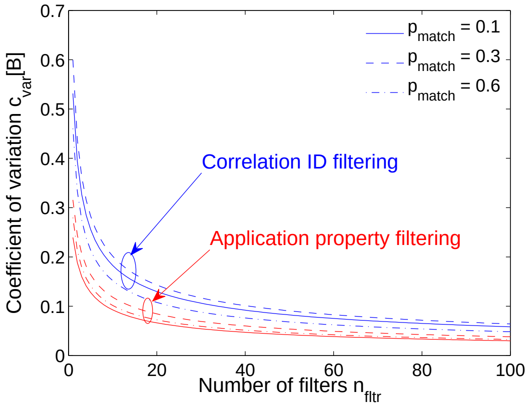
The model parameters can be calculated vice-versa by $n_{\text{fltr}} = \frac{E[R^2]}{E[R]}$ and $p_{\text{match}} = \frac{E[R]}{n_{\text{fltr}}}$. Furthermore, the third moment is

$$E[R^3] = \frac{E[R^2]^2}{E[R]} . \tag{5.41}$$

We are interested in the coefficient of variation $c_{\text{var}}[B]$ of the message service time based on a message replication grade, which is distributed according to this scaled Bernoulli distribution. This is calculated using Equations (5.36), (5.33), and (5.34). Figure 5.8(a) shows $c_{\text{var}}[B]$ depending on the number of filters n_{fltr}, the match probability p_{match}, and the filter type. The coefficient of variation $c_{\text{var}}[B]$ converges for an increasing number of filters to values that depend on p_{match} and the filter type. The coefficient of variation is at most $c_{\text{var}}[B] = 0.65$ and we cannot find any larger values for any other parameters of p_{match}.



(a) Replication grade R distributed according to a scaled Bernoulli distribution.



(b) Replication grade R distributed according to a Binomial distribution.

Figure 5.8: *Impact of n_{fltr} and p_{match} on the coefficient of variation $c_{var}[B]$ of the message processing time B.*

Binomial Distribution: The scaled Bernoulli distribution is probably not realistic enough to model the distribution of the message replication grade. Now, we assume that the filters n_{fltr} match messages independently of each other with a probability of p_{match}. Then, the resulting replication grade follows a Binomial distribution:

$$P(R = k) = \binom{n_{\text{fltr}}}{k} \cdot p_{\text{match}}^{k} \cdot (1 - p_{\text{match}})^{n_{\text{fltr}}-k}. \tag{5.42}$$

Furthermore, the second and third moments [130] are

$$E[R^2] = n_{\text{fltr}} \cdot p_{\text{match}} \cdot (1 - p_{\text{match}}) \,, \tag{5.43}$$

$$E[R^3] = E[R]^2 - E[R^2] - E[R] \cdot E[R^2] + 2 \cdot \frac{E[R^2]^2}{E[R]} \,. \tag{5.44}$$

We conduct the same study like above and observe in Figure 5.8(b) that the coefficient of variation $c_{\text{var}}[B]$ decreases quickly for an increasing number of filters n_{fltr} to values between 0.064 and 0.033, depending on the filter type.

After all, the second moment of the service time is bound by Equation (5.34) and the second moment of the replication grade, cf. Equations (5.37), (5.40), and (5.43), respectively. Realistic coefficients of variations of the message service time lie between 0.0 and 0.2 and coefficients larger than 0.65 are impossible. Therefore, we work in the following exemplarily with the values 0.0, 0.2, and 0.4 because they cover realistic values in the considered scenarios.

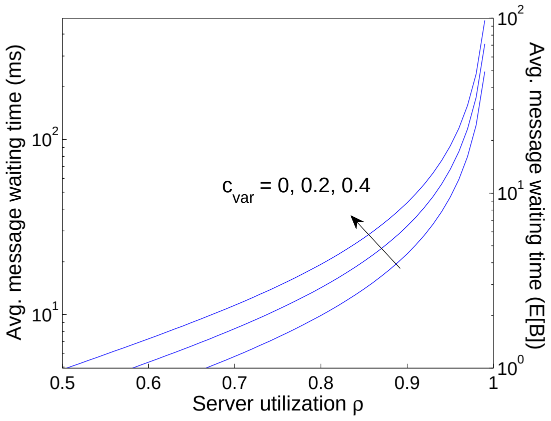


Figure 5.9: *Impact of the server utilization ρ and the coefficient of variation $c_{\text{var}}[B]$ of the message service time on the average message waiting time $E[W]$.*

5.2.3 Analysis of the Message Waiting Time

After evaluating a model for the message waiting time and its properties and limits, we present the analytical results for this model in the following section.

Average Message Waiting Time

The average message waiting time at the JMS server can be calculated using Equation (5.2). Figure 5.9 depicts this depending on the server utilization ρ in a specific application scenario with $n_{\text{fltr}} = 100$ application property filters and a constant replication grade of $R = 1$. The left y-axis shows the corresponding waiting time in ms. It is obvious that the average waiting time $E[W]$ increases with ρ. We can generalize the result by indicating the waiting time as a multiple of the average message processing time $E[B]$ on the right y-axis, which also

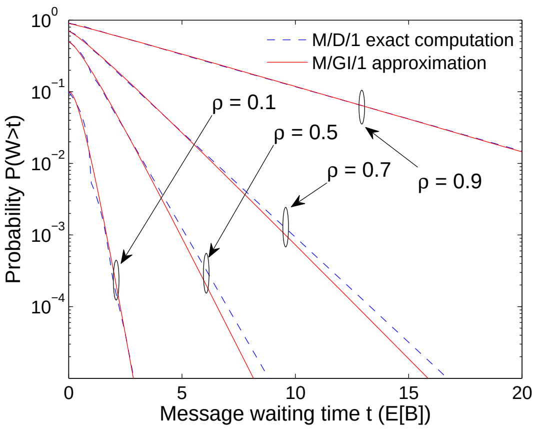


Figure 5.10: *Comparison of the exact and approximated CCDF for the waiting time of an $M/D/1-\infty$ queuing system for different utilization values ρ.*

approximates the mean queue length in packets. Based on this normalized y-axis, we can easily compare the average message waiting time $E[W]$ from different application scenarios that have different means $E[B]$ and coefficients of variations $c_{\text{var}}[B]$. Figure 5.9 illustrates that the mean waiting time is sensitive to the coefficient of variation of the message processing time B and that it increases with $c_{\text{var}}[B]$. Note that the normalized diagram in Figure 5.9 provides also a lookup table for the average message waiting time $E[W]$ in any application scenario with a matching coefficient of variation $c_{\text{var}}[B]$.

Message Waiting Time Distribution

In addition to the mean waiting time, we are also interested in the entire distribution function. According to $M/GI/1-\infty$ results, the waiting time probability for a message is $p_w = \rho$. With Equations (5.2) and (5.3) we know the first and second moment of the message waiting time such that we can calculate the first and second moment of the waiting time W_{wc} regarding only delayed customers by

$$E[W_{wc}] \quad = \quad \frac{E[W]}{\rho}, \qquad E[W_{wc}^2] = \frac{E[W^2]}{\rho} \, . \tag{5.45}$$

The Gamma-distribution has a positive range and can be viewed as the continuation of the exponential and Erlang-distribution for coefficients of variations different from $c_{\text{var}}[X] = \frac{1}{\sqrt{k}}$, $k \in \mathbb{N}$ [127]. We approximate the waiting time distribution of the delayed calls $P(W_{wc} \leq t)$ by fitting their two parameters $\alpha = \frac{1}{c_{\text{var}}[W_{wc}]}$ and $\beta = \frac{E[W_{wc}]}{\alpha}$. Thus, we get the waiting time distribution function regarding all calls by

$$P(W \leq t) \quad = \quad (1-\rho) + \rho \cdot P(W_{wc} \leq t). \tag{5.46}$$

This Gamma-approximation is exact for an exponentially distributed service time and leads to very good approximation results for other service time distributions as we have shown in [14].

To be aware of the error for other service time distributions, we consider an $M/D/1-\infty$ system for which the exact waiting time distribution can be calculated [131]. However, numerical instabilities arise such that the corresponding probabilities can be calculated only for small time values [146]. Figure 5.10 shows a comparison of the exact and the approximated CCDFs of waiting time for $M/D/1-\infty$ for various system utilization values ρ. The approximation is good enough to sufficiently estimate exact delay quantiles, in particular for large values of the utilization ρ. As the approximation works well for $c_{\text{var}}[B]=0$ and $c_{\text{var}}[B]=1$, we use it also in the following for $0 < c_{\text{var}}[B] < 1$.

Figure 5.11 shows the CCDF of the message waiting time W for a server utilization of $\rho = 0.9$ and for a coefficient of variation of $c_{\text{var}}[B] = 0.0$, 0.2, and 0.4 on a normalized (to $E[B]$) x-axis. The distribution functions are clearly shifted towards larger waiting time values with increasing $c_{\text{var}}[B]$, which is

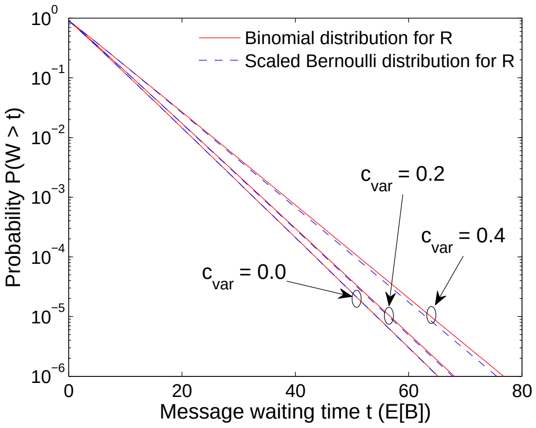


Figure 5.11: *Impact of the coefficient of variation $c_{\text{var}}[B]$ and the distribution type of the message replication grade R on the CCDF of the message waiting time W for a server utilization of $\rho = 0.9$.*

consistent with the results obtained in Section 5.2.3. The deterministic, the scaled Bernoulli, and the Binomial distribution coincide for $c_{\text{var}}[B] = 0$ and thus lead to the same waiting time distribution of the messages. Furthermore, we can hardly see any difference between the waiting time distribution function for the binomial and the Bernoulli distribution of the replication grade R. Thus, we can neglect the exact distribution type of the message service time and work with its first two moments instead. In the following, we assume a messages service time based on a binomially distributed message replication grade R.

Message Waiting Time Quantile

The p-quantile or p-percentile $Q_p[W]$ specifies the lowest duration for which $P(W \leq Q_p[W]) \geq p$ holds. It says "$p \cdot 100\%$ of all messages wait shorter than $Q_p[W]$" and yields thereby a "quasi upper bound" on W if p is large. Figure 5.12

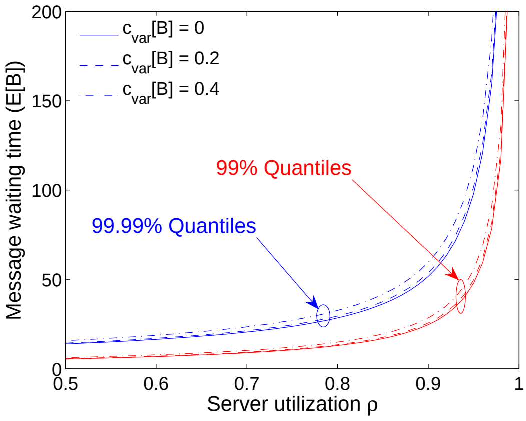


Figure 5.12: *Impact of the server utilization ρ and the coefficient of variation $c_{\text{var}}[B]$ of the message service time on the 99% and 99.99% quantiles of the message waiting time.*

shows the 99% and 99.99% quantile of the waiting time on a normalized y-axis depending on the server utilization ρ and the coefficient of variation $c_{\text{var}}[B]$ of the message service time. The 99.99% quantile of the waiting time is substantially larger than the 99% quantile. The quantiles increase with the server utilization ρ and they are substantially larger than the means of the waiting time $E[W]$ in Figure 5.9. The impact of the coefficient of variation $c_{\text{var}}[B]$ is notable but the impact of the server utilization ρ is much larger. If the probability for immediate processing $1-p_w(\rho)$ is significantly smaller than the quantile value p, the quantiles $Q_p[W]$ are significantly larger than the corresponding means $E[W]$ (cf. Figure 5.9) and they are more sensitive to the coefficient of variation $c_{\text{var}}[B]$ of the message service time B.

If we limit the server utilization to $\rho = 0.9$, the message waiting time is less than $50 \cdot E[B]$, i.e., a waiting time of $50 \cdot E[B]$ is not exceeded with a probability of 99.99%. With that probability a maximum waiting time of at most 1 s is guaranteed as long as $E[B]$ is smaller than 20 ms. For this time limit and an average replication grade of $E[R] = 1$ for the above scenario, up to 1369 or 2845 filters may be installed on the JMS server for application property or correlation ID filtering, respectively. However, in this case, the maximum server capacity is only $\lambda^{\text{max}}_{\rho=0.9} = 45$ messages per second which is very low. Hence, for a system designer the message waiting time can be neglected as long as 1 s is acceptable for a server utilization of $\rho = 0.9$ or less. Thus, if a sufficiently high throughput is achieved, the waiting time is small.

5.3 Performance Comparison of Distributed JMS Server Architectures

The capacity of a JMS server is bounded by the performance of its CPU. If it does not suffice to support a certain message rate from n publishers to m subscribers, a distributed architecture might be useful to alleviate the problem. We consider two basically different simple architectures: *publisher-side JMS server replication* (PSR) and *subscriber-side JMS server replication* (SSR).

5.3.1 Publisher-Side JMS Server Replication

With PSR, each publisher has its own local JMS server, for which subscribers can register. The concept is visualized in Figure 5.13. Each publisher-side $M/GI/1$–∞ system supports a message rate λ_i and their average message replication grade is $E[R_i]$. Since the messages are filtered already at the publishers, the traffic load imposed on the network interconnecting publishers and subscribers is $\sum_{0 \leq i < n} \lambda_i \cdot E[R_i]$.

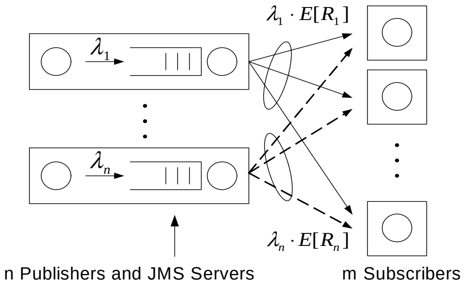


Figure 5.13: *Publisher-side JMS server replication (PSR).*

A drawback of this distributed PSR architecture is the fact that all subscribers have to register in parallel for n JMS servers at distributed publisher sites instead

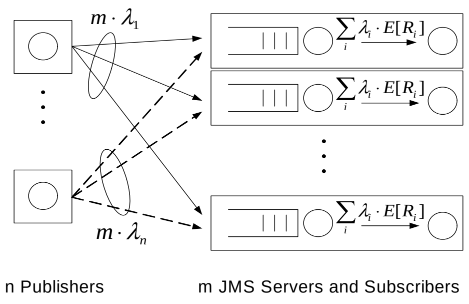

$m \cdot \lambda_1$
$m \cdot \lambda_n$
$\sum_i \lambda_i \cdot E[R_i]$
n Publishers
m JMS Servers and Subscribers

Figure 5.14: *Subscriber-side JMS server replication (SSR).*

of a single one. This disturbs the elegant communication interface of JMS over a single server. Thus, additional entities must be introduced to allow a transparent communication like with a single server, but this is not scope of this work.

5.3.2 Subscriber-Side JMS Server Replication (SSR)

With SSR, each subscriber has its own JMS server for which publishers can register. The concept is visualized in Figure 5.14. Since the messages are filtered only at the subscribers, the message rate for each subscriber-side $M/GI/1-\infty$ system is $\lambda = \sum_{1 \leq i \leq n} \lambda_i$. Thus, the overall traffic carried in the network is $m \cdot \lambda$.

Since m is an upper bound on R_i, SSR produces significantly more traffic in the network than PSR. Like with PSR, the elegant communication interface of JMS is also compromised by the SSR architecture because every publisher needs to multicast its messages to all JMS servers at m different subscriber sites instead of to a single one. However, this problem is not our present concern.

5.3.3 Capacity Comparison of PSR and SSR

For the performance comparison of both architectures, we consider the following scenario. All nodes have the same computation power. In particular, we assume that they have the same capacity as the machines in our experiments in Chapter 3 because our numerical study relies on the values t_{rcv}, t_{fltr}, and t_{tx} that were obtained for these machines. Furthermore, the message rates λ_i of all publishers are equal and the average replication grades $E[R_i]$ for their messages are the same such that we can denote them uniformly by $E[R]$. In addition, each subscriber has $n_{\text{fltr}} = 10$ different filters.

For PSR, the capacity of the distributed JMS system

$$\lambda_{PSR}^{\text{max}} = n \cdot \min_{1 \leq i \leq n} (\lambda_i^{max}) \tag{5.47}$$

is the n-fold multiple of the minimum of all individual JMS server capacities λ_i^{max}. Similarly to Equation (5.31), it can be calculated under the above stated assumptions by

$$\lambda_{PSR}^{\text{max}} = \rho \cdot n \cdot \left(t_{\text{rcv}} + m \cdot n_{\text{fltr}} \cdot t_{\text{fltr}} + E[R] \cdot t_{\text{tx}}\right)^{-1} . \tag{5.48}$$

Thus, the system capacity depends on n and m and is thereby application scenario specific. In case of subscriber-side JMS server replication, the capacity of the distributed JMS system $\lambda^{\text{max}} = \min_{0 \leq i < m}(\lambda_i^{max})$ is the minimum of all individual JMS server capacities λ_i^{max}. It can be calculated under the above stated assumptions by

$$\lambda_{SSR}^{\text{max}} = \rho \cdot \left(t_{\text{rcv}} + n_{\text{fltr}} \cdot t_{\text{fltr}} + E[R] \cdot t_{\text{tx}}\right)^{-1} . \tag{5.49}$$

In contrast to λ_{PSR}^{max}, the expression for λ_{SSR}^{max} is independent of n and m.

Figure 5.15 illustrates the impact of the parameters n and m on the capacities $\lambda_{PSR}^{\text{max}}$ and $\lambda_{SSR}^{\text{max}}$ of both distributed JMS systems. The results are calculated for an average replication grade of $E[R] = 1$, a maximum server utilization of $\rho = 0.9$, and correlation ID filtering. The capacity $\lambda_{SSR}^{\text{max}}$ for SSR yields a horizontal line since it is independent of the parameters n and m. The capacity for PSR increases linearly with n and decreases almost reciprocally for large values of m. PSR outperforms SSR for medium or large values of n and for small or medium

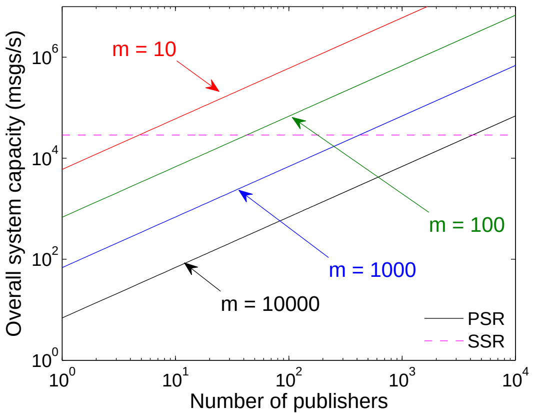


Figure 5.15: *Capacity comparison of PSR and SSR scenario for a server utilization of $\rho = 0.9$, an average replication grade of $E[R] = 1$, and m subscribers.*

values of m. Note that a large m can reduce the capacity of a single JMS server so much that waiting time problems arise. For example, for $m = 10^4$ and a large n the distributed system has still a large capacity but the capacity of a single publisher-side server is only 7 msgs/s leading to average waiting times of 1 s and to 99.99% quantiles of 10 s. We get similar results for correlation ID filtering.

The capacity lines in Figure 5.15 intersect where both Equations (5.48) and (5.49) yield the same results. Thus, we conclude that PSR outperforms SSR if the following inequality holds

$$n \quad > \quad \frac{t_{\text{rcv}} + m \cdot n_{\text{fltr}} \cdot t_{\text{fltr}} + E[R] \cdot t_{\text{tx}}}{t_{\text{rcv}} + n_{\text{fltr}} \cdot t_{\text{fltr}} + E[R] \cdot t_{\text{tx}}}. \tag{5.50}$$

It gives a recommendation under which circumstances PSR or SSR should be implemented to cope with a large number of publishers or subscribers.

In conclusion, PSR achieves system capacity scalability with respect to an increasing number of publishers, but the capacity degrades with an increasing number of subscribers. In contrast, SSR provides system capacity scalability for an increasing number of subscribers but its capacity does not scale with an increasing number of publishers. Since both architectures represent an artificial bound of the system architecture, the real system design considers something in between, as done in practical systems, by clustering JMS server machines. The performance evaluation of such a system is out-of-scope of this work. Although, the methodology and models presented in this section can be used as a first step for evaluating JMS servers arranged in a cluster based architecture.

5.4 Concluding Remarks

The overall focus of the previous chapter is to analyze the queuing behavior of a JMS server in addition to our models based on mean values for the message throughput. While developing the system models we consider a real-time scenario which imposes time constraints on the numerical evaluation of our prediction model. We fitted an $M/GI/1-\infty$ queuing system for calculating the message waiting times of the FioranoMQ JMS server.

However, there are no explicit expressions except for some special cases. The waiting time distribution function can only be obtained by a numerical inversion of its Laplace-Stieltjes transform which is not numerically feasible in our scenario. We identified also other numerical approximations as not appropriate for our scenario. Thus, we proposed the Gamma-approximation that estimates the waiting time distribution function by a Gamma-distribution based on the first three moments of the service time distribution. We showed the accuracy of the Gamma-approximation and proved that it is very accurate even for the calculation of 99.999% quantiles of the waiting times. Overall, the Gamma-approximation is not restricted to our scenario, but can be used in general for technical control systems where fast numerical results for the waiting times are necessary.

We observed from our experimental analytical parameter study in Chapter 4 that the values for both the message processing time and the corresponding server capacity λ^{max} in messages per second range over several orders of magnitude depending on the application scenario. Both additional filters and unnecessarily sent messages reduce the JMS server capacity. Thus, we studied this phenomenon and gave a recommendation for the configuration of subscribers to maximize the JMS server capacity based on the message match probability of filters. We modelled the JMS server by an $M/GI/1-\infty$ queuing system and presented three different distributions for the message replication grade, which lead to a significantly different variability of the message processing time. We showed that the average message waiting time is mainly influenced by the server utilization. The sensitivity analysis showed that the processing time variability plays only a marginal role. We used a normalized diagram which can be used as a lookup table for various application scenarios. The 99.999% quantile gives a “quasi upper bound” on the waiting time and an estimate on the required buffer space at the JMS server. Finally, we concluded that extensive waiting times do not occur as long as the server is not overloaded and as long as its throughput is medium or high. These results are of general nature and are also valid for other servers than the FioranoMQ.

Finally, we introduced two distributed JMS server architectures: publisher- and subscriber-side server replication (PSR, SSR). We compared the capacity of both alternatives by the use of our simple throughput model. The capacity $\lambda^{\text{max}}_{PSR}$ of PSR scales well for an increasing number of publishers and the capacity $\lambda^{\text{max}}_{SSR}$ of SSR scales well enough for an increasing number of subscribers. However, none of them scales well for both requirements. We provided some recommendations for the usage of PSR or SSR in order to design a real system. The decision which approach is closer to the requirements by the environment depends on the application scenario.

6 Conclusions and Future Trends

In modern software architectures, efficient application-to-application communication is an important requirement. Introducing a *message-oriented middleware* (MOM) decouples the communicating partners by providing standard interfaces. This enables a flexible exchange of information even in heterogeneous application environments.

In this thesis we investigated a messaging infrastructure as a central entity in the application communication process. Such a messaging infrastructure may become a bottleneck of the overall system. Therefore, we evaluated the scalability in terms of the overall message throughput. There are several approaches available, which support an efficient communication of applications. We picked the *publish/subscribe* communication pattern as a widely accepted architecture which fulfills a large set of requirements of a modern application design, like one-to-many communication and filtering of information according to the subscribers' interests. A key feature is the decoupling of the publisher's and subscriber's communication. Many approaches are available to design and optimize a publish/subscribe system. However, in practice, a variety of different implementations and combinations of filtering mechanisms are possible, which increases the complexity for predicting a resulting system performance. We evaluated the limits of different messaging systems and identified the most impacting parameters in order to give hints for dimensioning and designing a messaging system regarding different application scenarios.

Thus, we designed a measurement-based methodology to identify critical parameters and evaluated them for different realizations of the *Java mes-*

saging service (JMS). Also, a categorization of the different parameters is given according to their system impact regarding server utilization, message throughput, subscriber utilization, and information granularity. The proposed measurement method assesses the software capacity of the server with respect to the maximum achievable message throughput. To that end, we sent messages to the server in a saturated manner by a test tool. The presented numerical results are valid for the testbed and the JMS server configurations used in our environment. Using different hardware components, especially current and future hardware architectures, e.g., multi-core processor systems and optimized caching strategies, may cause different behaviors in scalability. A parallel execution of processes introduces additional overhead to the code in terms of synchronizing each single thread. Also an optimal configuration of each JMS server based on its software capabilities can improve its overall performance. Since these optimizations typically consider a specific application scenario, a generalized performance prediction is limited. Therefore, we used the default configuration for all considered vendors, since it typically is optimized to the "common case". The categorization of critical parameters and the described experimental methodology is in general a good approach to evaluate a messaging based system in varying application scenarios. Using our method we conducted a series of measurements regarding the number of connected clients and the message size. For the number of clients, we found that a sufficiently high number of publishers or subscribers is required to fully utilize the server machine and to obtain a typical message throughput.

Considering this result, it can be concluded that the maximum system performance can only be achieved if a sufficiently large number of clients are connected to the messaging system. In case of information selection, we tested the coarse-grained design option using topics and message header-based filtering. Using header-based filtering, we can also differ the fixed header part which is more efficient for all measured servers but is limited in its applicability. Filtering for the application header properties has increased flexibility, but – depending on the filter complexity – impacts the throughput performance significantly. For instance, the FioranoMQ servers message throughput performance decreased most with only activating simple filters. All other tested servers experience a minor impact by simply activating filters. The impact increases if we introduce complex filter scenarios where a filter consists of multiple components connected by a logical operator like “OR” or “AND”. The considered parameters in our measurements have a different level of impact depending on the server type and the application scenario. Therefore, parameters should be evaluated during the design phase of the development of a messaging application.

In addition, we tested three different options for the same application scenario, using simple filters, complex OR-filters, and IN-filters. The IN-filter operator is the best option in terms of throughput performance but cannot be modified while active. The simple filter option has the lowest performance but is very flexible for updating the filter parameters. Hence, during the design of an application, the filter-type has to be chosen carefully. The presented measurements focus on the binary transmission of a message payload and header-based filtering. In current and future systems we notice also an increasing number of content-based filtering approaches, e.g., based on an XML encoded message payload. Considering such an environment, the expressiveness of a single filter and as a consequence its complexity might increase dramatically whereas the performance suffers from extracting information out of the message payload. For messaging-systems with a large message throughput we recommend, based on our results, a split of the message payload and a header containing meta information. The JMS framework, considered in this work, supports only the header-based filtering approach.

We further considered the impact of network transport protocols and tested the impact of different connection and session aggregation levels of subscriptions. We found that there is only a minor impact while choosing grouped TCP-connections or JMS session aggregation. A major impact can be observed during setup of the initial connections to a JMS server. This is for example the case in a busy-hour scenario, where a flash-crowd of clients tries to connect to the server at the same time. Considering the Bea WebLogic server, it prefers incoming subscriptions and therefore stalls delivering messages, which might lead to an overload scenario. The ActiveMQ server tends to prefer message delivery which results in large subscription registration times and a possible message loss with newly connected subscribers. Thus, we suggest to test the impact of network and subscription handling in the desired application scenario.

In order to dimension an application environment based on a messaging infrastructure, it is important to predict the required message throughput performance of the message-oriented middleware system. Therefore, we proposed a measurement-based approach to derive simple prediction models regarding the impact of filtering and the replication grade. We were able to fit a base model, which we adapted to special behaving servers, like the SunMQ or the WebSphereMQ. Using a linear regression method we calculated system-specific parameters from measurements. We were able to validate our results and observed a good match of the results of the model and our measurement results. We also enhanced the models to cover complex filtering scenarios. Thus, the proposed mathematical approximation models for the message processing time are accurate enough to predict the message throughput for our application scenarios. The proposed models consider the average message replication grade, the overall number of installed filters, and the number of different filters. From our results we can conclude that all server types have a basically different performance behavior, but we found individual models for each server following our measurement based approach. Finally, we illustrated the applicability of our performance models by applying our models to four simple application scenarios.

Since the message throughput prediction models are based on average values for the message processing time and the replication grade, we introduced an $M/GI/1-\infty$ queuing system to estimate the message waiting times for the overall system. This enables the prediction of the server load. Thus, if we try to predict and avoid failure scenarios, a very fast calculation of the numerical values of the model is desirable. Since no numerical approximation method is efficient enough to calculate these numbers sufficiently fast, we proposed our so-called *Gamma-approximation* which is based on the Gamma-distribution. The computing effort needed to apply this approximation method is negligible and therefore, the best choice for our scenario. The approximation is also useful in all other scenarios where a fast and accurate approximation of the numerical values of an $M/GI/1-\infty$ queuing system is necessary. Since the message processing time and the corresponding message throughput capacity of the server depends heavily on the application scenario, we gave a recommendation for the configuration of subscribers to maximize the JMS server capacity based on the probability a filter matches. We showed that the average message waiting time is mainly influenced by the server utilization. In contrast, the sensitivity analysis showed that the processing time variability has only a minor impact. Finally, we concluded that extensive waiting times do not occur as long as the server is not overloaded and as long as its throughput is medium or high.

Since the messaging system represents a central entity, scalability regarding the overall message throughput can be achieved by distributing the server logic. Therefore, we compared two diverse design options, the *publisher-side server replication* (PSR) and the *subscriber-side server replication* (SSR) considering our performance models. We found that the choice of the best architecture heavily depends on the application scenario, but our models can be used to identify the best choice. To support the choice we gave recommendations for the usage of PSR or SSR.

In the course of the monograph, we showed that the careful design of an application-to-application messaging infrastructure has several application-specific impact parameters. Thus, we identified the limits and the typical performance values of such an environment by several measurement series. We developed a basic and an enhanced model to predict the message throughput performance regarding the publish/subscribe model and using the JMS framework. The performance models consider the joint impact of filtering and replication grade. Also deviations of the mean values can be considered by applying our queuing model. Following our approach, it is possible to determine the critical parameters regarding the desired scenario and get an estimate for the necessary equipment needed for the deployment in an application scenario.

Future Trends

The presented methods and models are also useful in future system scenarios. The scenarios presented in this work covered only a part of the possible application scenarios. Our scenarios are sufficient to show the generic behavior of a messaging system and its performance limits. However, if we consider stock trading applications, for example, the message throughput and the time constraints may be more mission critical and have to rely on faster equipment and components. To cope with this problem, some vendors try to implement the messaging infrastructure in hardware. This is a realistic future scenario, if reconfigurable hardware is powerful enough to be flexible in terms of its possible communication interfaces and the achievable message throughput.

In addition, the application of the publish/subscribe communication pattern to lower network layers, as proposed by the PSIRP project [103], is an interesting approach which requires a careful design of the architecture and the used algorithms. Although we focused on the JMS framework, our models can be extended and applied to other frameworks and application scenarios.

A clear trend on application-layer can be seen in combining application-layer communication protocols considering messaging. Currently, a large group of vendors and open source developers decided to combine their efforts to extend the advanced message queuing protocol (AMQP) as an open standard. This enables a flexible combination of different messaging approaches by using a standardized underlying protocol.

Several other models consider already our performance models presented in this work. Since we focused on the applicability of our approach, the effort to adapt them to another application scenario and different requirements is low. Summarizing, messaging based on the publish/subscribe communication pattern is an efficient and flexible communication architecture for the design of future applications.

Bibliography of the Author

— Journals —

[1] M. Menth, R. Henjes, S. Gehrsitz, and C. Zepfel, “Throughput Performance of Popular JMS Servers”, *ACM SIGMETRICS Performance Evaluation Review (PER)*, Vol. 34(1), 2006.

[2] A. Binzenhöfer, G. Kunzmann, and R. Henjes, “Design and Analysis of a Scalable Algorithm to Monitor Chord-based P2P Systems at Runtime”, *Concurrency and Computation: Practice and Experience – Special Issue on HotP2P 06*, 2007.

[3] R. Henjes, M. Menth, and V. Himmler, “Throughput Performance of the BEA WebLogic JMS Server”, *International Transactions on Systems Science and Applications*, Vol. 3, Number 3, 2007.

— Conference Papers —

[4] T. Koulouris, R. Henjes, K. Tutschku, and H. de Meer, “Implementation of Adaptive Control for P2P Overlays”, in *Proceedings Fifth Annual International Working Conference on Active Networks, Dec. 10–12 2003*, Kyoto, Japan, 2003.

[5] A. Binzenhöfer, D. Staehle, and R. Henjes, “On the Fly Estimation of the Peer Population in a Chord-based P2P System”, in *19th International Teletraffic Congress (ITC19)*, Beijing, China, 2005.

[6] A. Binzenhöfer, D. Staehle, and R. Henjes, “On the Stability of Chord-based P2P Systems”, in *GLOBECOM 2005*, St. Louis, MO, USA, 2005.

[7] G. Kunzmann, A. Binzenhöfer, and R. Henjes, “Analyzing and Modifying Chord’s Stabilization Algorithm to Handle High Churn Rates”, in *MICC & ICON 2005*, Kuala Lumpur, Malaysia, 2005.

[8] A. Binzenhöfer, G. Kunzmann, and R. Henjes, “A Scalable Algorithm to Monitor Chord-based P2P Systems at Runtime”, in *Third International Workshop on Hot Topics in Peer-to-Peer Systems (Hot-P2P) in conjunction with the IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece, 2006.

[9] R. Henjes, M. Menth, and S. Gehrsitz, “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.

[10] R. Henjes, M. Menth, and C. Zepfel, “Throughput Performance of Java Messaging Services Using Sun Java System Message Queue”, in *High Performance Computing & Simulation Conference (HPC&S)*, Bonn, Germany, 2006.

[11] R. Henjes, M. Menth, and C. Zepfel, “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.

[12] M. Menth and R. Henjes, “Analysis of the Message Waiting Time for the FioranoMQ JMS Server”, in 26^{th}*IEEE International Conference on Distributed Computing Systems (ICDCS)*, Lisbon, Portugal, 2006.

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

[14] M. Menth, R. Henjes, C. Zepfel, and P. Tran-Gia, “Gamma-Approximation for the Waiting Time Distribution Function of the $M/G/1-\infty$ Queue”, in 2^{nd}*Conference on Next Generation Internet Design and Engineering*, Valencia, Spain, 2006.

[15] R. Henjes, M. Menth, and S. Gehrsitz, “Throughput Performance of ActiveMQ JMS Server”, in 15^{th}*Kommunikation in Verteilten Systemen (KiVS)*, Bern, Switzerland, 2007.

[16] R. Henjes, M. Menth, and V. Himmler, “Impact of Complex Filters on the Message Throughput of the ActiveMQ JMS Server”, in *International Teletraffic Congress (ITC)*, Ottawa, Canada, 2007.

[17] D. Schwerdel, D. Günther, R. Henjes, B. Reuther, and P. Müller, “German-Lab Experimental Facility”, in *3rd Future Internet Symposium 2010*, Berlin, Germany, 2010.

General References

[18] M. H. Ackroyd, “Computing the Waiting Time Distribution for the G/G/1 Queue by Signal Processing Methods”, *IEEE Transactions on Communications*, Vol. 28, No. 1, 1980.

[19] M. H. Ackroyd, “Stationary and Cyclostationary Finite Buffer Behaviour Computation via Levinson’s Method”, *AT&T Bell Laboratories Technical Journal*, Vol. 63, No. 10, 1984.

[20] P. Tran-Gia, “Zeitdiskrete Analyse verkehrstheoretischer Modelle in Rechner- und Kommunikationssystemen”, *46. Bericht über verkehrstheoretische Arbeiten, Universität Stuttgart*, 1988.

[21] P. Tran-Gia and H. Ahmadi, "Analysis of a Discrete-Time $G^X/D/1-S$ Queueing System With Applications in Packet-Switching Systems", in *IEEE Infocom*, New Orleans, 1988.

[22] International Organization for Standardization, "Information Technology - Database Languages - SQL." ISO/IEC 9075, 1992.

[23] P. Tran-Gia and R. Dittmann, "A Discrete-Time Analysis of the Cyclic Reservation Multiple Access Protocol", *Performance Evaluation*, Vol. 16, 1992.

[24] J. Abate, G. L. Choudhury, and W. Whitt, "Calculation of the GI/G/1 Waiting Time Distribution an its Cumulants from Pollaczek's Formulas", *International Journal of Electronics and Communications*, Vol. 47, No. 5/6, 1993.

[25] P. A. Bernstein, "Middleware: An Archictecture for Distributed System Services", Tech. Rep. CRL 93/6, Cambridge MA, 1993.

[26] P. Tran-Gia, "Discrete-Time Analysis Technique and Application to Usage Parameter Control Modeling in ATM Systems", in *8th Australian Teletraffic Research Seminar*, Melbourne / Australia, 1993.

[27] J. Abate, G. L. Choudhury, and W. Whitt, "Exponential Approximations for Tail Probabilities in Queues I: Waiting Times", *Operations Research*, Vol. 43, No. 5, 1995.

[28] J. Abate and W. Whitt, "Numerical Inversion of Laplace Transforms of Probability Distributions", *INFORMS Journal on Computing*, Vol. 7, No. 1, 1995.

[29] B. Segall and D. Arnold, "Elvin has Left the Building: A Publish/Subscribe Notification Service with Quenching", in *Proceedings of AUUG 97*, Brisbane, Australia, 1997.

[30] A. Carzaniga, *Architectures for an Event Notification Service Scalable to Wide-Area Networks*. PhD thesis, Milano, Italy: Politecnico di Milano, 1998.

[31] R. Strom, G. Banavar, T. Chandra, M. Kaplan, K. Miller, B. Mukherjee, D. Sturman, and M. Ward, "Gryphon: An Information Flow Based Approach to Message Brokering", in *International Symposium on Software Reliability Engineering*, 1998.

[32] J. Abate and W. Whitt, "Explicit M/G/1 Waiting Time Distributions for a Class of Long-Tail Service Time Distributions", *Operations Research*, Vol. 25, No. 1, 1999.

[33] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra, "Matching Events in a Content-Based Subscription System", in *Proceedings of the* 18^{th} *Annual ACM Symposium on Principles of Distributed Computing*, ACM, 1999.

[34] A. Carzaniga, D. Rosenblum, and A. Wolf, "Achieving Scalability and Expressiveness in an Internet-scale Event Notification Service", in *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, ACM, 2000.

[35] M. Day, J. Rosenberg, and H. Sugano, "A Model for Presence and Instant Messaging", *Request for Comments*, Vol. 2778, 2000.

[36] R. Gummadi and B. Hohlt, "Efficient Implementation of a Publish-Subscribe-Notify Model Using Highly-Concurrent B-Trees", Tech. Rep., 2000.

[37] A. Milewski and T. Smith, "Providing Presence Cues to Telephone Users", in *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, ACM, 2000.

[38] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, “Content Based Routing with Elvin4”, in *Proceedings of AUUG 00*, 2000.

[39] A. Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, “Efficient Filtering in Publish-Subscribe Systems using Binary Decision Diagrams”, in *Proceedings of the* 23^{rd} *International Conference on Software Engineering*, Washington, DC, USA, IEEE Computer Society, 2001.

[40] G. Cugola, E. Di Nitto, and A. Fuggetta, “The JEDI Event-Based Infrastructure and its Application to the Development of the OPSS WFMS”, *IEEE Transactions on Software Engineering*, 2001.

[41] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, “Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems”, *ACM SIGMOD 2001*, Vol. 30, 2001.

[42] G. Mühl, “Generic Constraints for Content-Based Publish/Subscribe”, in 9^{th} *International Conference on Cooperative Information Systems (CoopIS)*, London, UK, 2001.

[43] J. Pereira, F. Fabret, F. Llirbat, and H. A. Jacobsen, “WebFilter: A High-throughput XML-based Publish and Subscribe System”, in *Proceedings of the* 27^{th} *International Conference on Very Large Data Bases*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2001.

[44] J. W. Roberts, “Traffic Theory and the Internet”, *IEEE Communications Magazine*, Vol. 1, No. 3, 2001.

[45] A. Carzaniga and A. L. Wolf, “A Benchmark Suite for Distributed Publish/Subscribe Systems”, Tech. Rep. CU-CS-927-02, Software Engineering Research Laboratory, Department of Computer Science, University of Colorado, Boulder, Colorado, USA, 2002.

[46] G. Fox and S. Pallickara, “The Narada Event Brokering System: Overview and Extensions”, in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2002.

[47] H. K. Y. Leung, “Subject Space: A State-Persistent Model for Publish/Subscribe Systems”, in *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*, 2002.

[48] G. Mühl, L. Fiege, and A. Buchmann, “Filter Similarities in Content-based Publish/Subscribe Systems”, in *Conference on Architecture of Computing Systems (ARCS)*, Springer, 2002.

[49] G. Mühl, *Large-Scale Content-Based Publish/Subscribe Systems*. PhD thesis, Darmstadt University of Technology, 2002.

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

[51] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf, “Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems”, in *Proceedings of the 35^{th} Hawaii International Conference on System Sciences*, 2002.

[52] T. Wolf, “Benchmark für EJB-Transaction und Message-Services”, Master’s thesis, Universität Oldenburg, 2002.

[53] R. Baldoni, M. Contenti, and A. Virgillito, “The Evolution of Publish/Subscribe Communication Systems”, in *Future Directions in Distributed Computing*, Springer, 2003.

[54] R. Baldoni, M. Contenti, S. T. Piergiovanni, and A. Virgillito, “Modelling Publish/Subscribe Communication Systems: Towards a Formal

Approach", in 8^{th} *International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*, 2003.

[55] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola, "Introducing Reliability in Content-Based Publish-Subscribe through Epidemic Algorithms", in *International Workshop on Distributed Event-Based Sytems (DEBS)*, 2003.

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

[57] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The Many Faces of Publish/Subscribe", *ACM Computing Surveys (CSUR)*, Vol. 35, No. 2, 2003.

[58] I. Gorton, J. Almquist, N. Cramer, J. Haack, and M. Hoza, "An Efficient, Scalable Content-Based Messaging System", in *EDOC '03: Proceedings of the 7^{th} International Conference on Enterprise Distributed Object Computing*, Washington, DC, USA, IEEE Computer Society, 2003.

[59] T.-Y. Hsiao, M.-C. Cheng, H.-T. Chiao, and S.-M. Yuan, "FJM: A High Performance Java Message Library", *IEEE International Conference on Cluster Computing*, 2003.

[60] H. Leung and H. Jacobsen, "Efficient Matching for State-Persistent Publish/Subscribe Systems", in *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*, IBM Press, 2003.

[61] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet", *ACM SIGCOMM Computer Communication Review*, Vol. 33, No. 1, 2003.

[62] D. Tam, R. Azimi, and H.-A. Jacobsen, "Building Content-Based Publish/Subscribe Systems with Distributed Hash Tables", in *Proceedings of*

the International Workshop On Databases, Information Systems and Peer-to-Peer Computing, 2003.

[63] L. Zanolin, C. Ghezzi, and L. Baresi, "An Approach to Model and Validate Publish/Subscribe Architectures", in *Proceedings of the SAVCBS*, Vol. 3, Helsinki, Finland, 2003.

[64] J. Cantarella and M. Piatek, "Tsnnls: A Solver for Large Sparse Least Squares Problems with Non-Negative Variables", *ArXiv Computer Science e-prints*, Vol. cs.MS/0408029, 2004.

[65] F. Cao and J. P. Singh, "Efficient Event Routing in Content-Based Publish/Subscribe Service Network", in *Proceedings of the* 23^{rd} *Conference of the IEEE Communications Society (INFOCOM'04)*, 2004.

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

[67] P. Costa, M. Migliavacca, G. P. Picco, and G. Cugola, "Epidemic Algorithms for Reliable Content-Based Publish-Subscribe: An Evaluation", in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2004.

[68] U. Farooq, E. W. Parsons, and S. Majumdar, "Performance of Publish/Subscribe Middleware in Mobile Wireless Networks", *ACM SIGSOFT Software Engineering Notes*, Vol. 29, No. 1, 2004.

[69] L. Fiege, A. Zeidler, A. Buchmann, R. Kilian-Kehr, and G. Mühl, "Security Aspects in Publish/Subscribe Systems", in *Proceedings of the* 3^{rd} *Intl Workshop on Distributed Event-Based Systems (DEBS)*, IEE The Institution of Electrical Engineers, 2004.

[70] T.-H. Kaoa, C.-P. Hung, H.-T. Chiao, T.-Y. Hsiao, Y.-S. Chang, and S.-M. Yuan, "A Fast Java Message System based-on IP Multicast", Tech. Rep.,

Department of Computer and Information Science National Chiao Tung University Hsinchu, Taiwan R.O.C., 2004.

[71] Krissoft Solutions, “JMS Performance Comparison”, Tech. Rep., Krissoft Solutions, 2004. Available at `http://www.fiorano.com/comp-analysis/jms_perf_comp.htm`.

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

[73] M. Menth, “A Framework for Modelling and Solving Discrete Finite Markov Chains”, Technical Report, No. 326, University of Würzburg, Institute of Computer Science, 2004.

[74] S. Oh, S. L. Pallickara, S. Ko, J.-H. Kim, and G. Fox, “Cost Model and Adaptive Scheme for Publish/Subscribe Systems on Mobile Environments”, in *Second International Workshop on Active and Programmable Grids Architectures and Components*, 2004.

[75] P. R. Pietzuch, *Hermes: A Scalable Event-Based Middleware*. PhD thesis, Queens’ College, University of Cambridge, 2004.

[76] J. F. Shortle, P. H. Brill, M. J. Fischer, D. Gross, and D. M. B. Masi, “An Algorithm to Compute the Waiting Time Distribution for the M/G/1 Queue”, *INFORMS Journal on Computing*, Vol. 16, No. 2, 2004.

[77] P. Triantafillou and I. Aekaterinidis, “Content-based Publish-Subscribe Over Structured P2P Networks”, in *Proceedings of the Third International Workshop on Distributed Event-Based Systems (DEBS)*, 2004.

[78] Y. Zhao, D. Sturman, and S. Bhola, “Subscription Propagation in Highly-Available Publish/Subscribe Middleware”, in *Proceedings of the* 5^{th} *ACM/IFIP/USENIX International Conference on Middleware*, New York, NY, USA, Springer-Verlag New York, Inc., 2004.

[79] R. Baldoni, R. Beraldi, S. T. Piergiovanni, and A. Virgillito, "On the Modelling of Publish/Subscribe Communication Systems", *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 12, 2005.

[80] S. Bittner and A. Hinze, "A Detailed Investigation of Memory Requirements for Publish/Subscribe Filtering Algorithms", in *International Conference on Cooperative Information Systems (CoopIS)*, Agia Napa, Cyprus, 2005.

[81] S. Kale, E. Hazan, F. Cao, and J. Singh, "Analysis and Algorithms for Content-based Event Matching", in *Proceedings of the* 25^{th} *IEEE International Conference on Distributed Computing Systems Workshops*, 2005.

[82] V. Muthusamy, *Infrastructureless Data Dissemination: A Distributed Hash Table Based Publish/Subscribe System*. PhD thesis, University of Toronto, 2005.

[83] M. Antollini, M. Cilia, and A. Buchmann, "Implementing a High Level Pub/Sub Layer for Enterprise Information Systems", in *Proceedings of the* 8^{th} *International Conference on Enterprise Information Systems: Databases and Information Systems Integration*, Paphos, Cyprus, 2006.

[84] S. Behnel, L. Fiege, and G. Mühl, "On Quality-of-Service and Publish-Subscribe", in *Proceedings of the* 26^{th} *IEEE International Conference on Distributed Computing Systems Workshops (ICDCS)*, 2006.

[85] A. Corsaro, L. Querzoni, S. Scipioni, S. Piergiovanni, and A. Virgillito, "Quality of Service in Publish/Subscribe Middleware", in *Global Data Management* (R. Baldoni and G. Cortese, eds.), IOS Press, 2006.

[86] V. Ramasubramanian, R. Peterson, and E. Sirer, "Corona: A High Performance Publish-Subscribe system for the World Wide Web", in *Proceedings of Networked System Design and Implementation (NSDI)*, 2006.

[87] S. Tarkoma, *Efficient Content-based Routing, Mobility-aware Topologies, and Temporal Subspace Matching*. PhD thesis, University of Helsinki, Finland, 2006.

[88] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "Constructing scalable overlays for pub-sub with many topics", in *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, New York, NY, USA, ACM, 2007.

[89] G. Chockler, R. Melamed, Y. Tock, and R. Vitenberg, "SpiderCast: a Scalable Interest-Aware Overlay for Topic-Based Pub/Sub Communication", in *Proceedings of the Inaugural International Conference on Distributed Event-Based Systems (DEBS)*, New York, NY, USA, ACM, 2007.

[90] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann, "Workload Characterization of the SPECjms2007 Benchmark", *Formal Methods and Stochastic Models for Performance Evaluation*, 2007.

[91] K. Sachs, S. Kounev, M. Carter, and A. Buchmann, "Designing a Workload Scenario for Benchmarking Message-oriented Middleware", in *Proceedings of the 2007 SPEC Benchmark Workshop*, SPEC, 2007.

[92] The OpenJMS Group, "OpenJMS", 2007. Available at `http://openjms.sourceforge.net/`.

[93] M. Campanella, "Federated E-infrastructure Dedicated to European Researchers Innovating in Computing Network Architectures", *Future Internet Conference*, 2008.

[94] N. Laranjeiro, M. Vieira, and H. Madeira, "Experimental Robustness Evaluation of JMS Middleware", in *IEEE International Conference on Services Computing (SCC)*, Vol. 1, 2008.

[95] H. Subramoni, G. Marsh, S. Narravula, P. Lai, and D. Panda, "Design and Evaluation of Benchmarks for Financial Applications using Advanced Message Queuing Protocol (AMQP) over InfiniBand", in *Workshop on High Performance Computational Finance (In conjunction with SC '08)*, Austin, TX, 2008.

[96] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line Speed Publish/Subscribe Inter-Networking", *ACM SIGCOMM Computer Communication Review*, Vol. 39, No. 4, 2009.

[97] J. Kramer, "Advanced Message Queuing Protocol (AMQP)", *Linux Journal*, No. 187, 2009.

[98] J. Martins and S. Duarte, "Routing Algorithms for Content-based Publish/Subscribe Systems", *IEEE Communications Tutorials and Surveys*, 2009.

[99] OW2 Consortium, "JORAM: Java^{TM} Open Reliable Asynchronous Messaging", 2009. Available at `http://joram.ow2.org/`.

[100] Red Hat, Inc., "JBossMQ", 2009. Available at `http://community.jboss.org/wiki/JBossMQ`.

[101] K. Sachs, S. Kounev, S. Appel, and A. Buchmann, "Benchmarking of Message-oriented Middleware", in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, ACM, 2009.

[102] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann, "Performance Evaluation of Message-oriented Middleware using the SPECjms2007 Benchmark", *Performance Evaluation*, Vol. 66, No. 8, 2009.

[103] S. Tarkoma, M. Ain, and K. Visala, "The Publish/Subscribe Internet Routing Paradigm (PSIRP): Designing the Future Internet Architecture",

in *Towards the Future Internet – A European Research Perspective* (G. Tselentis, J. Domingue, A. Galis, A. Gavras, D. Hausheer, S. Krco, V. Lotz, and T. Zahariadis, eds.), IOS Press, 2009.

[104] AMQP Working Group, "Advanced Message Queuing Protocol", 2010. Available at `http://www.amqp.org`.

[105] D. Crockford, "JavaScript Object Notation (JSON)", 2010. Available at `http://www.json.org/`.

[106] H. Liu, *Management of Uncertainties in Publish/Subscribe System*. PhD thesis, Department of Computer Science, University of Toronto, 2010.

[107] Oracle Corporation, "Bea WebLogic Application Server", 2010. Available at `http://www.oracle.com/bea/index.html`.

[108] SpringSource, "RabbitMQ", 2010. Available at `http://www.rabbitmq.com`.

[109] Wikipedia Community, "Java Message Service", 2010. Available at `http://en.wikipedia.org/wiki/Java_Message_Service`.

[110] A. Zahemszky, B. Gajic, C. Rothenberg, C. Reason, D. Trossen, D. Lagutin, J. Tuononen, and K. Katsaros, "Experimentally-Driven Research in Publish/Subscribe Information-Centric Inter-Networking", in *Proceedings of the 6^{th} International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, 2010.

[111] Åke Björck, *Numerical Methods for Least Squares Problems*. SIAM Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[112] Apache Foundation, *ActiveMQ, Reference Documentation*, 2010. Available at `http://activemq.apache.org/`.

[113] Apache Incubator, *ActiveMQ, JMeter Performance Test Tool*, 2006. Available at `http://www.activemq.org/jmeter-performance-tests.html`.

[114] Bea Systems, *Bea WebLogic Server 9.0*, 2006. Available at `http://dev2dev.bea.com`.

[115] T. Erl, *Service-Oriented Architecture: Concepts, Ttechnology, and Design*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2005.

[116] Fiorano Software, Inc., *FioranoMQ Documentation Center*, 2005. Available at `http://www.fiorano.com/devzone/doc_fmq.htm`.

[117] Fiorano Software, Inc., *FioranoMQ^{TM}: Meeting the Needs of Technology and Business*, 2004. Available at `http://www.fiorano.com/whitepapers/whitepapers\_fmq.pdf`.

[118] Fiorano Software, Inc., *Illustrating the FioranoMQ^{TM} performance advantage over SonicMQ, Tibco EMS, Jboss Messaging, Sun JAVA MQ, ActiveMQ and IBM WebSphere MQ*, 2005. Available at `http://www.fiorano.com/whitepapers/`.

[119] S. Godard, *Sysstat Monitoring Utilities 5.1.4*, 2004. Available at `http://perso.wanadoo.fr/sebastien.godard/`.

[120] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Haase, *Java Message Service API Tutorial and Reference: Messaging for the J2EE Platform*. Addison-Wesley, 2002.

[121] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

[122] IBM Corporation, *IBM WebSphere MQ 6.0*, 2005. Available at `http://www-01.ibm.com/software/integration/wmq/`.

[123] IBM Hursley, *Performance Harness for Java Message Service*, 2005. Available at `http://www.alphaworks.ibm.com/tech/perfharness`.

[124] iMatix, Corp., *ZeroMQ*, 2010. Available at `http://www.zeromq.org/`.

[125] JBoss, *JBoss JMS New Performance Benchmark*, 2006. Available at `http://community.jboss.org/docs/DOC-10476`.

[126] L. Kleinrock, *Queueing Systems*, Vol. 1: Theory. New York: John Wiley & Sons, 1^{st} ed., 1975.

[127] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill, 3^{rd} ed., 2000.

[128] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. No. 15 in Classics in Applied Mathematics, Society for Industrial & Applied Mathematics, 1995.

[129] Oracle Corporation, *Java Message Service API Rev. 1.1*, 2002. Available at `http://java.sun.com/products/jms/`.

[130] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill Book Company, 2^{nd} ed., 1984.

[131] N. U. Prabhu, *Queues and Inventories*. New York, London, Sydney: John Wiley & Sons, Inc., 1965.

[132] Progress Software, *Enterprise-Grade Messaging*, 2004. Available at `http://www.sonicsoftware.com/products/docs/sonicmq.pdf`.

[133] Progress Software, *Sonic Test Harness*, 2005. Available at `http://communities.progress.com/pcom/docs/DOC-29828`.

[134] Progress Software Corporation, *High Performance Messaging with JMS - A Benchmark Comparison of SonicMQ® vs. IBM MQSeries® 5.2*, 2001. Available at `http://www.sonicsoftware.com`.

[135] L. P. Seelen, H. C. Tijms, and M. H. Van Horn, *Tables for Multi-Server Queues*. North-Holland, 1985.

[136] B. Snyder, D. Bosanac, and R. Davies, *ActiveMQ in Action – Early Access Edition*. Manning Publications Co., 2010.

[137] Software Engineering Research Laboratory, *SIENA: Publish/Subscribe Wide-Area Event Notification*. University of Colorado, 2005. Available at `http://serl.cs.colorado.edu/~serl/siena/`.

[138] Sun Microsystems Inc., *Sun Java System Application Server 9.1 Developer's Guide*, 2008. Available at `http://docs.sun.com/app/docs/doc/819-3672/`.

[139] Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054, *Sun ONE Message Queue, Reference Documentation*, 2006. Available at `http://developers.sun.com/prodtech/msgqueue/`.

[140] H. Takagi, *Queueing Analysis Volume 1: Vacation and Priority Systems*. North-Holland, 1991.

[141] S. Terry and T. Shawn, *Enterprise JMS programming*. New York, NY, USA: John Wiley & Sons, Inc., 2002.

[142] The MathWorks, Inc., *MATLAB 7.0*, 2005. Available at `http://www.mathworks.com/products/matlab/`.

[143] Tibco Software, Inc., *TIBCO Enterprise Message Service*, 2004. Available at `http://www.tibco.com`.

[144] H. C. Tijms, *A First Course in Stochastic Models*. West Sussex, England: John Wiley & Sons, 2003.

[145] H. C. Tijms, *Stochastic Modelling and Analysis: A Computational Approach*. John Wiley & Sons, 1986.

[146] H. C. Tijms, *Stochastic Models: An Algorithmic Approach*. John Wiley & Sons, 1994.

[147] Tomlab Optimization, *The TOMLAB Optimization Environment*. Available at `http://tomlab.biz/`.

[148] Ziena Optimization, Inc., *KNITRO 4.0*, 2005. Available at `http://www.ziena.com/knitro.htm`.

ISSN 1432-8801