Evaluation of Technologies for Communication between Monitoring and Analysis Component in Kieker

Bachelor's Thesis

Jan Beye

September 26, 2013

Kiel University
Department of Computer Science
Software Engineering Group

Advised by: Prof. Dr. Wilhelm Hasselbring
M. Sc. Florian Fittkau
Dipl.-Inf. Jan Waller

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

_____

Abstract

The number of distributed software systems is increasing. This makes reliable and efficient communication more important then ever. In the past years and decades many different technologies for communication have been developed. The purpose of this thesis is to evaluate different technologies for the communication of the monitoring and the analysis component of the monitoring software Kieker. Four different technologies have been implemented and evaluated with the metrics throughput of records per time interval and the loss rate.
Two low level protocols, i.e., TCP and UDP have been implemented. The evaluation turned out that TCP is the most efficient technology for the communication. UDP did not fulfill the expectation to be the fastest communication technology and stays behind TCP. The other two protocols are high level protocols. These are a JMS solution on the example of RabbitMQ and RMI. The evaluation showed that the overhead of RMI is too heavy to be an efficient communication technology. JMS is more efficient then RMI but the overhead for the communication with the JMS provider is still to heavy to be really efficient.
The reliability is given by all technologies besides UDP which has a loss rate of six percent in the evaluation. This maybe acceptable in streaming and multimedia applications but for many uses cases this circumstance is not acceptable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Distributed software systems and cloud computing are getting more and more important in our days. A cloud in this context is not a data-storage cloud like ownCloud [*ownCloud*] or iCloud [*iCloud*]. By cloud computing we refer to, for instance, Amazon EC2 [*Amazon EC2*], Eucalyptus [*Eucalyptus*], or OpenStack [*OpenStack*]. Two of the major advantages are the flexibility and scalability of those systems. However, there are also challenges that have to be solved. One of them is the communication between the different components of the distributed software system in the cloud. Whether to use direct communication via TCP/IP or more high-level protocols like JMS [Hapner et al. 2002] or JMX [Process 2006] is an very important decision. This decision is important because it has a major impact on the performance of the application. Reliability becomes important in this context. Unreliable protocols can be faster than reliable because the overhead is much smaller, but how much messages are getting lost? And is the lost arguable? So this isn't only interesting for cloud computing and distributed systems, its interesting for all applications, which exchange data and communicate with other applications.
In our days, another important topic is application monitoring. This Monitoring provides a live view onto the software, which is necessary for reacting on load peaks. The gathered information can be used to start new cloud instances. Monitoring also gives an insight to the application and shows where potentials for improvements are and it can be used for reengineering of the systems, to determine the important parts of the application and how often a part of an application is called. The live view approach is followed by the ExplorViz project [Fittkau 2012]. This project tries to visualize traces from monitored applications in real time with different views.

1.2 Goals

The goal of the thesis is to evaluate the best suited technology for the communication between the monitoring and analysis component in Kieker [van Hoorn et al. 2012] in the cloud. Technologies in this context can be low-level protocols like TCP or UDP, but also more high-level technologies like JMS or RMI. The evaluation should figure out which technology provides the best ratio between throughput and reliability requirements.

G1 - Development of a Performance Evaluation Method and Matrix

The most important goal of this thesis is to develop a reliable method for performance evaluation for Kieker in the cloud. For this purpose, it is also very important to define a good matrix for the evaluation which fits to the context. Metrics for the evaluation will be throughput and reliability of the technology. In our context, throughput describes the received monitoring records in respect to a timing frame. Reliability means that all send packets are getting received.

G2 - Performance Evaluation

As the second goal the implemented technologies will be evaluated, based on the developed evaluation method and matrix.

G3 - Analysis of the Results

The last goal of the thesis will be the analysis of the results with statistical means [Georges et al. 2007]. After finishing the analysis, the results will be used to determine the best fitting technology for the communication between the monitoring and analysis component in Kieker in the cloud.

1.3 Document Structure

In Chapter 2 the foundations are presented. Chapter 3 describes the project in whose context this thesis is written. The next chapter provides details about the implementation. Afterwards Chapter 5 shows the details of the performance evaluation. That chapter is followed by a chapter which discusses related work. In the last chapter, the results will be summarized and a outlook is given.

Chapter 2

Foundations and Technologies

2.1 Kieker

Kieker [van Hoorn et al. 2012] is a Monitoring Software, which monitors the execution of methods in a program. It is also possible to monitor resources like the CPU Utilization with Kieker. Kieker is developed by the Software Engineering Group of the Department of Computer Science at the University of Kiel, Germany. It is used in research projects and by industrial company's. Kieker is licensed under the Apache License, Version 2.0., which allows everybody to use and modify Kieker.

Architecture

As Figure 2.1 shows, Kieker consists mainly of two components, i.e., monitoring and the analysis component. These components communicate with a writer and reader system, which can be extended as needed. Its possible to store the data for later analysis in a repository like a database or a file or send it directly to the analysis with, for instance, JMS. Records are used for the communication. This record can also be created as it is needed for the context.

Analysis

The analysis component follows the pipe and filter architectural style. This system is designed as a plugin system. The plugin system easily allows to connect a reader, which reads the monitored records, with the various filters. This filters again can be designed as needed for the analysis. A second important concept about the plugin system is that with the plugin system the filters can be plugged together. Therefore, it is possible to combine various analysis steps.

Monitoring

The main function of the monitoring component is to collect monitoring information with interception technologies like Aspectj [AspectJ]. The collection is done by probes. The collected data gets stored or send to analysis applications with writers. Again one important features is that everything can be extended as needed.

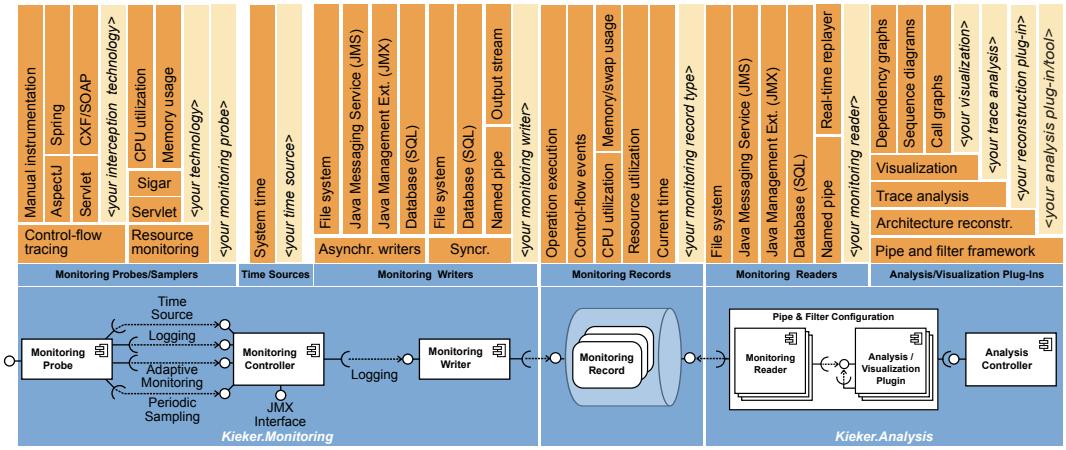


Figure 2.1. Kieker Overview
Source: `http://kieker-monitoring.net/features/`

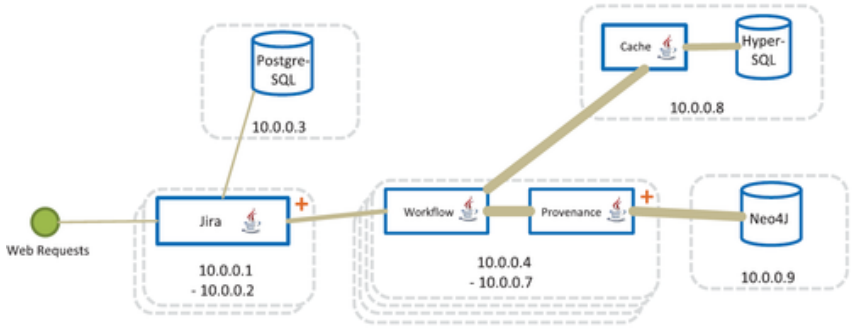
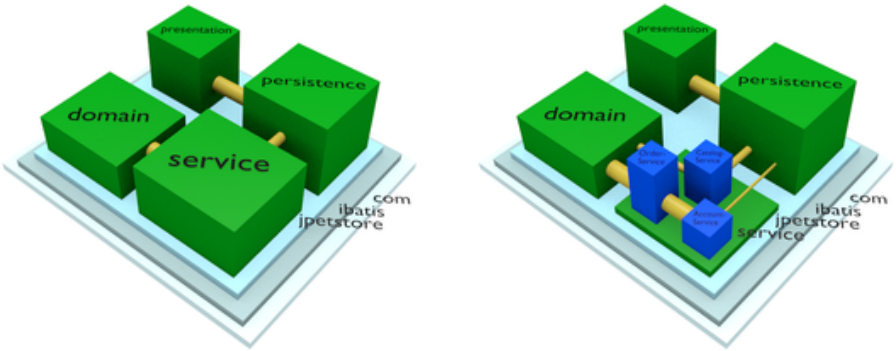


Figure 2.2. ExplorViz landscape view on the example of PubFlow
Source: `http://www.explorviz.net/`

2.2 ExplorViz

ExplorViz [Fittkau et al. 2013] is an approach to visualize applications in large software landscapes on demand and online. To visualize the application ExplorViz uses application monitoring, e.g., Kieker. To avoid the problem that most servers couldn't process such huge amount of monitoring records, which will be generated in large software landscapes, the work is distributed on worker nodes, for instance, with cloud computing. On these worker nodes a preprocessing of the monitoring data is done. During this preprocessing the records are consolidated to traces. To reduce the amount of data reductions techniques are used, e.g., equivalence classes. Afterwards the preprocessed data gets collected on a single node. The preprocessed data gets aggregated. The aggregated data and the



(a) System level view of the four components of the jPetStore

(b) System level view of the jPetStore with opend servcie component

Figure 2.3. ExplorViz system view on the example of jPetStore
Source: `http://www.explorviz.net/`

landscape model get transformed into a visualization model, which only includes the relevant informations. Since all this should take place in real time, it is very important that the communication between the nodes is efficient.
As visualization perspective two views are at the moment possible. An landscape view (see Figure 2.2) and a system view (see Figure 2.3). Figure 2.3a shows the four components of the jPetStore (`http://sourceforge.net/projects/ibatisjpetstore/`) and Figure 2.3b shows the relationships of the components with the opened service component. The landscape view is a 2D visualization and combines elements from the deployment and activity diagrams from the Unified Modeling Language (UML). The system view is a 3D visualization which uses the city metaphor for visualization. ExplorViz will be implemented as web application utilizing WebGL.

2.3 Benchmarking

Benchmarking describes the process of comparing different things which can be measured in the same unit. It is used in a lot of different sectors with different methods and goals, for example, for processes and products or in business economics. In this thesis we are using technology benchmarking. Technology benchmarking describes the comparison of similar or different technologies. It requires that all technologies have identical purpose. "In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. The term 'benchmark' is also mostly utilized for the purposes of elaborately-designed benchmarking programs

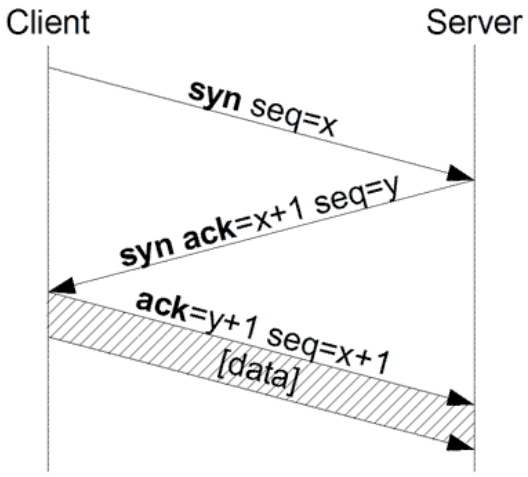


Figure 2.4. TCP handshake
Source: http://upload.wikimedia.org/wikipedia/commons/c/c7/300px-Tcp-handshake.png

themselves." As described in Kalibera and Jones [2013] to do an benchmarking with force of expression isn't easy. Modern software systems are quite complex and the execution is influenced by a lot of different factors, which can vary from run to run and aren't controllable. The difficulty is to find a methodology, which takes care of these problems and is repeatable. Advances in performance are often small in our field, which makes it very important to provide measurements of variation to eliminate the risk of measurement errors. This also makes the use of methods with statistical meaning very important. A lot of different benchmarks like the DaCapo benchmarks [Blackburn et al. 2006] have been developed to take care of all these problems. The concrete benchmarking design of this thesis is described in Chapter 5.

2.4 Communication Technologies

In the following, the four communication technologies for the evaluation are explained.

2.4.1 Transmission Control Protocol

The TCP is a transport protocol for communication in networks [Cerf et al. 1974]. It is a reliable protocol because as shown in Figure 2.4 the sender and the receiving server do a three way handshake to create a connection. During this handshake the sender sends an sequence number to the sever. This has to be acknowledged by the server. The acknowledgment of the server again has to be acknowledged by the receiver. Now the data can be transfered. Every transfered data has again to be acknowledged by the receiver.

This gives TCP the ability to react on damaged or lost data. The data transfer is done by sockets. On the network layer [ITU 1994] the internet protocol (IP) [Defense Advanced Research Projects Agency 1991] is used. TCP was developed in 1970 and became an industry-standard. Today, it is used in most networks.

2.4.2 User Datagram Protocol

The UDP is like TCP a transport protocol for communication in networks [RFC 1980]. In difference to TCP, UDP is an unreliable protocol. A sender just sends his data and gets no acknowledgment that the data was successfully received by the receiver, since no real connection is established between sender and receiver. The data packets are sent to the network with a destination and will be successfully received if something is listening at the destination. Otherwise the data will be lost. An important difference to TCP is that UDP can't deal with data in the wrong order. Therefore, the data must contain information about how it can be reconstructed when needed. Like in the case of TCP, UDP uses sockets for the communication and IP for the network layer. UDP had been developed in 1980 after TCP was marked as to heavy for multimedia streaming. Still UDP is mostly used for streaming, e.g., music streaming, because it is faster and the reliability is less important in such applications.

2.4.3 Java Message Service

JMS is developed under the Java Community Process and has the identification number JSR914 [Hapner et al. 2002]. It was developed for communication of distributed systems in a loosely coupled way. It guarantees reliability and the messages are exchanged asynchronously. As shown in Figure 2.5 for JMS a provider\server and one or more producers and one or more consumers are required [Basic JMS API Concepts]. JMS supports two main messaging concepts. The Point-to-Point Messaging Domain concept visualized in Figure 2.6 and the Publish/Subscribe Messaging Domain concept shown in Figure 2.7. In both cases the producer sends his message to the queue and has no knowledge about the receivers. The receiver also has no knowledge about the sender. Only the exchange format is known from both sides.
In the Point-to-Point concept the consumers consume the messages actively from the queue. After consumption, the consumer sends an acknowledgment. This concept is followed by, for instance, RabbitMQ [*RabbitMQ*]
In the Publish/Subscribe concept the message is sent to a topic. The consumers must register at the provider for the topic. The incoming messages will be delivered from the provider to the consumer. This concept is followed by, for instance, ActiveMQ [*ActiveMQ*].
JMS is not bound to a single protocol. Two common protocols for JMS are Advanced Message Queuing Protocol (AQMP) and TCP.

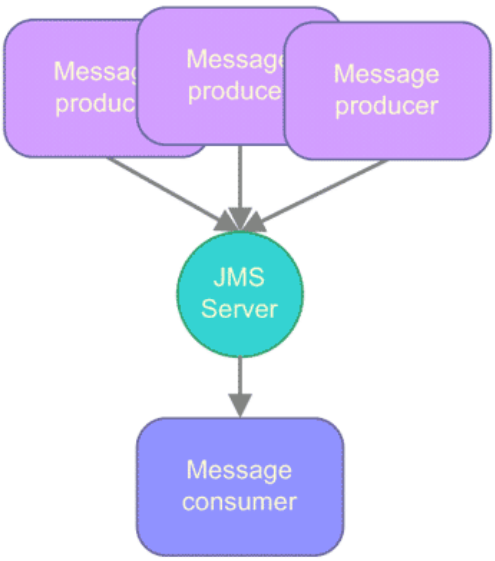


Figure 2.5. JMS overview
Source: `http://software.intel.com/en-us/articles/bitter-messages-java-messaging-anti-patterns`

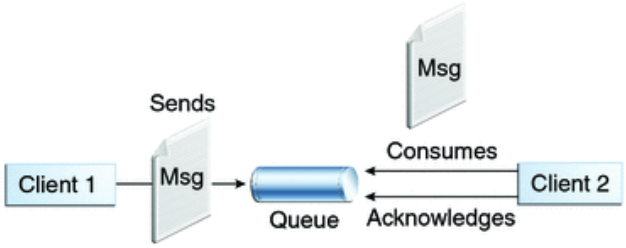


Figure 2.6. JMS point to point
Source: `http://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html`

2.4.4 Java Remote Method Invocation

RMI is based on client-server communication [*Java Remote Method Invocation*]. The server registers an object at the RMI-Registry. This is shown in step one in Figure 2.8. Now the object can be look up by the client in the RMI-Registry, as in step two of the figure. The RMI-Registry returns the server stub on a lookup, this is done in step three. The communication is established between the sever stub on the client side and the server skeleton on the server side as shown in step four. The client can call all methods on the object like it would be a local object. In difference to real local calls and objects, exceptions concerning the network communication must be handled by the method calls. In the call,

Figure 2.7. JMS publish subscribe
Source: `http://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html`

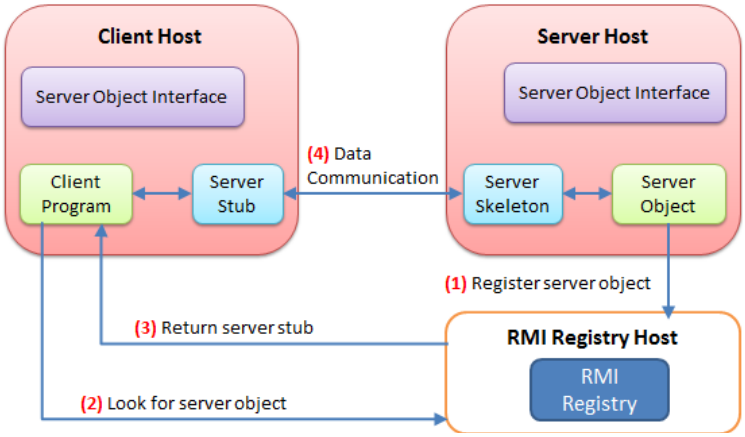


Figure 2.8. RMI overview
Source: `http://lycog.com/wp-content/uploads/2011/03/java-rmi-overview.png`

all necessary data can be put and transfered to the server. After receiving the call the server can process the data provided with the call. In comparison to the other technologies, RMI has the ability to provide informations with the return value. This makes an direct feedback possibly.

Chapter 3

Project Module - “Kieker in the Cloud”

The challenge of the project module - “Kieker in the Cloud”, was to bring the monitoring software Kieker into the cloud [Rohr et al. 2008]. One requirement had been that the analysis of the monitored data is done in real time on nodes in the cloud. Another requirement had been to develop a software which takes care to scale the nodes for the analysis and to distribute the monitoring to the analysis.

3.1 Kieker Architecture and Requirement Analysis

The first step of the project started with an analysis of the architecture of Kieker and the requirements. The architecture analysis was done by reverse engineering of the compiled jar file and some research in the source code of Kieker. We figured out that Kieker, as shown in Figure 3.1, consists of four packages.

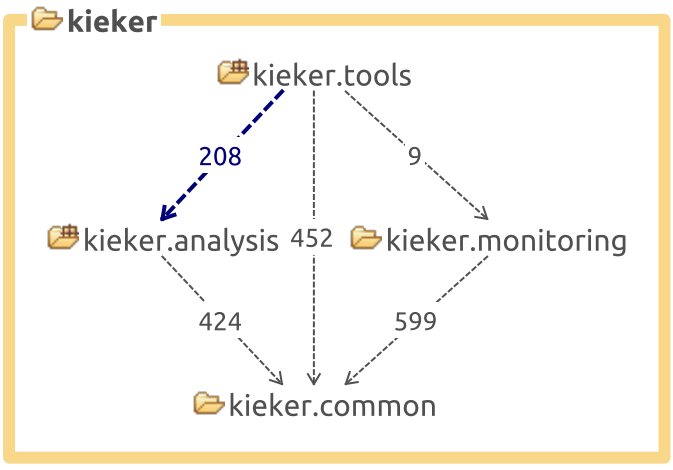


Figure 3.1. Kieker reverse engineered

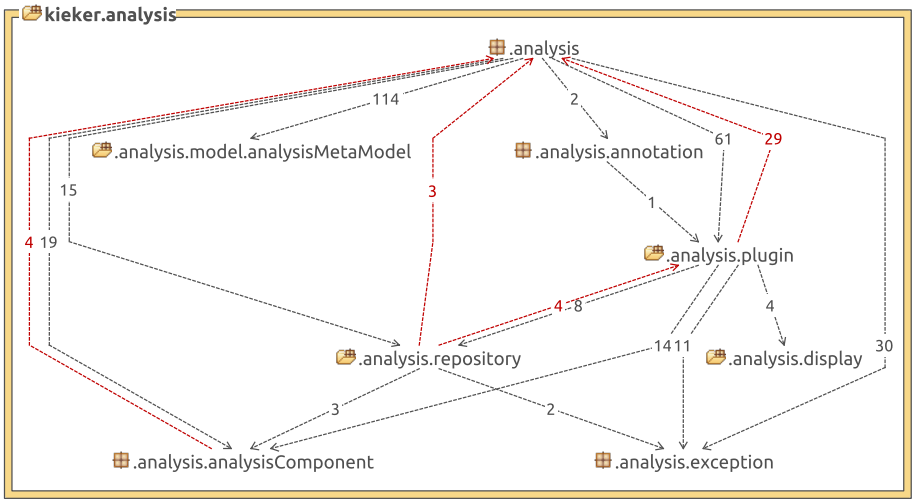


Figure 3.2. Packages in the Kieker analysis component

The two main packages are the Analysis package (Figure 3.2) and the Monitoring package (Figure 3.3). These two packages contain the main functionality for recording the monitoring data and analyze them. The Common package (Figure 3.4) is a helper package, which contains the different records and some other functionality needed by the Analysis and the Monitoring package. The last package, the Tools package (Figure 3.5) contains some useful tools like an player to replay logs.

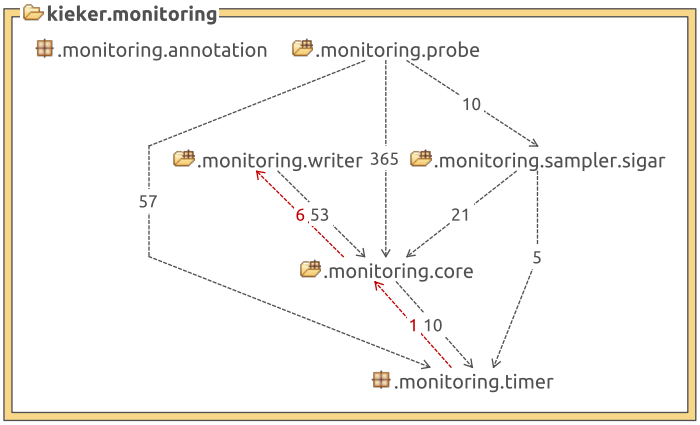


Figure 3.3. Packages in the Kieker monitoring component

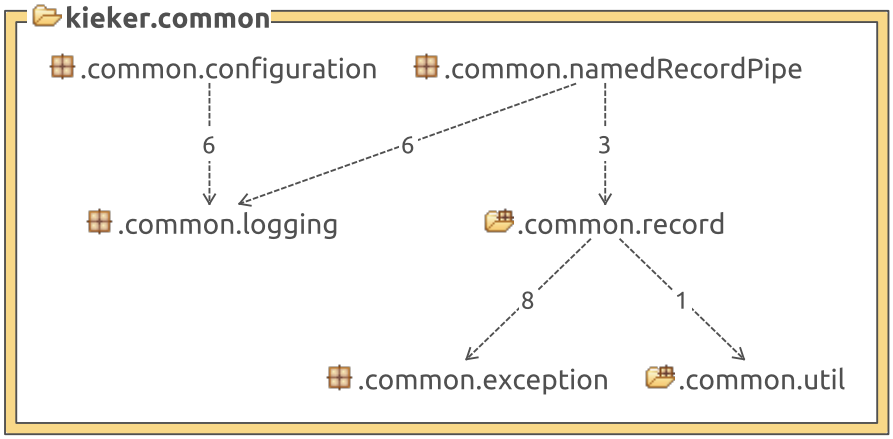


Figure 3.4. Packages in the Kieker common component

In the requirement analysis we discovered that we needed to write a new writer and a corresponding reader to send the monitored data to the analysis. For the project we decided to use RabbitMQ. From this requirement the question come up "Which technology provides the best performance for the communication between the monitoring and anylsis component of Kieker in the cloud?", which will be answered by this thesis. As the next requirement we defined, that we needed a tool what is responsible for the scaling of the analysis nodes. We first considered to extend the tool SLAstic [van Hoorn et al. 2009] to fulfill our requirements. After an analysis of the tool we decided to build a new tool based on SLAstic, called SLAstic Lite. We decided to split the analysis into multiple analysis workers and a single master, the installation is shown in Figure 3.6. All analysis nodes have there own RabbitMQ server. The analysis worker do a filtering of the monitored data to find similarities. This part is marked with the blue lines in Figure 3.6. Afterwards the analyzed data gets send to the master, where the data is collected and gets again analyzed for similarity with the data of the other nodes. This part is marked with the yellow lines in Figure 3.6. For the analysis we decided that we needed new filters. The new filters had to be compatible with JBPM work flows [*JBPM*], which are, for example, used in the Pubflow Project [*Pubflow*]. The last requirement we defined, was that we needed resource monitoring on the analysis nodes to give SLAstic Lite the information, which are necessary for scaling the number of cloud instances. The way of the cpu utilization monitoring is shown with the red lines in Figure 3.6.

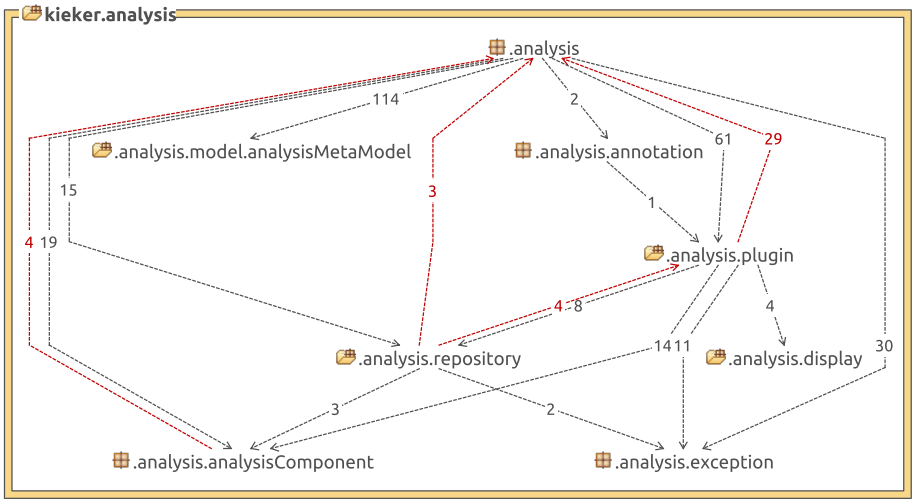


Figure 3.5. Packages in the Kieker tools component

3.2 Cloud

As the cloud system we used OpenStack in the version Grizzly from spring 2013. OpenStack is designed as a three component system as shown in Figure 3.7. The first component is the *Controller node*, which is responsible for the administration and hosts the web interface. This node contains all tools for user and project management, upload of images, and administration. The second node is the *Network node*, which handles the virtual network for the instances and the access from or to public networks. The external access has to be configured explicitly and is forbidden by default. The third node is the *Compute node*. This node is responsible for the virtualization of the images. OpenStack uses for this the libvirt virtualization, which is an open source virtualization api applicable for many virtualization technologies. OpenStack is designed as a service oriented software. All components serve as a service. There are different linux operating systems supported as host. For the project an ubuntu distribution was used. Its also possible to combine the components from the different nodes on a single node. The Controller and Network node had to be combined on a single node for the project. Its possible to add as many severs for compute nodes as needed. After the cloud has been installed, an image needs to be created. This image gets executed in the cloud. Its possible to create your own by using libvirt for creation, there are many tutorials on the Internet or an complete image can be downloaded from the Internet. The image can be modified as needed. After finishing the installation the image has to be added to the cloud. This can be done by command line or with the web interface.
For the project one image with a script which starts the analysis or the monitoring software

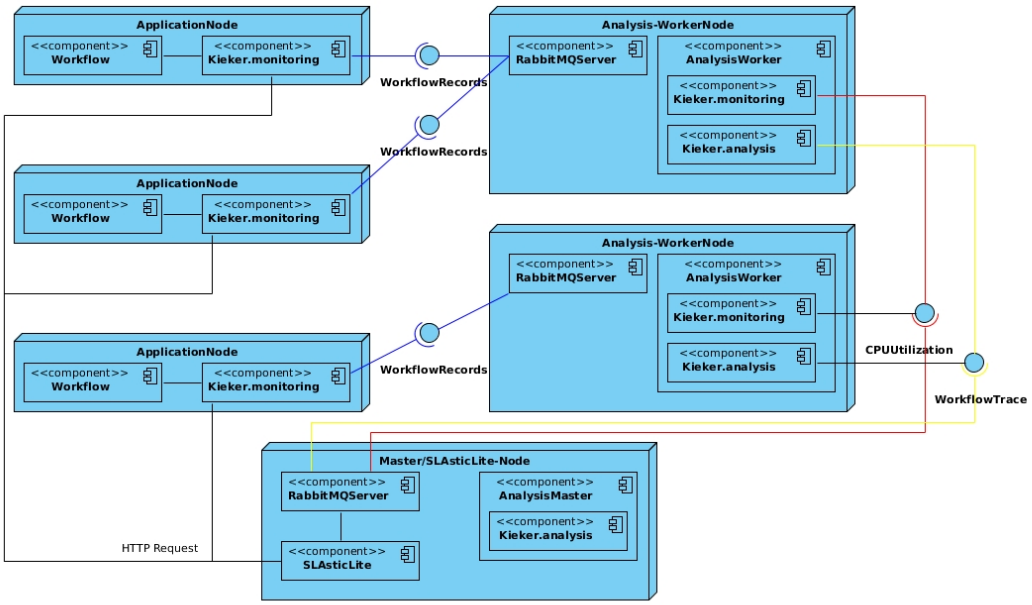


Figure 3.6. Deployment diagram of Kieker in the cloud

had been designed. This made changes easier because through the development only one image had to be administrated.

3.3 Implementation

To bring Kieker to the cloud it was necessary to develop an analysis application and on demand communication for Kieker. For the communication a RabbitMQ based solution for Kieker was implemented. The implementation included a reader for the analysis and a writer for the monitoring. Details of the implementation will be described in Chapter 4. For the analysis two standalone jars have been developed. This jars consist only of a controller, a reader and the filters needed for the analysis. The first jar was designed for the preprocessing on the analysis nodes of the cloud, the second was designed as the master analysis jar. The jars are configurable by parameters at start or by configuration files.
To analyze the given jars consisting of JBPM work flows, new filters had to be developed. The two main filters are reconstruction filters. A partial trace reconstruction filter for aggregation of partial traces and a trace reconstruction filter for complete traces. The filters are configurable. The main parameter is a time interval for the time frame in which the data is collected.
For the scaling of the number of cloud instances, SLAstic Lite had to be developed.

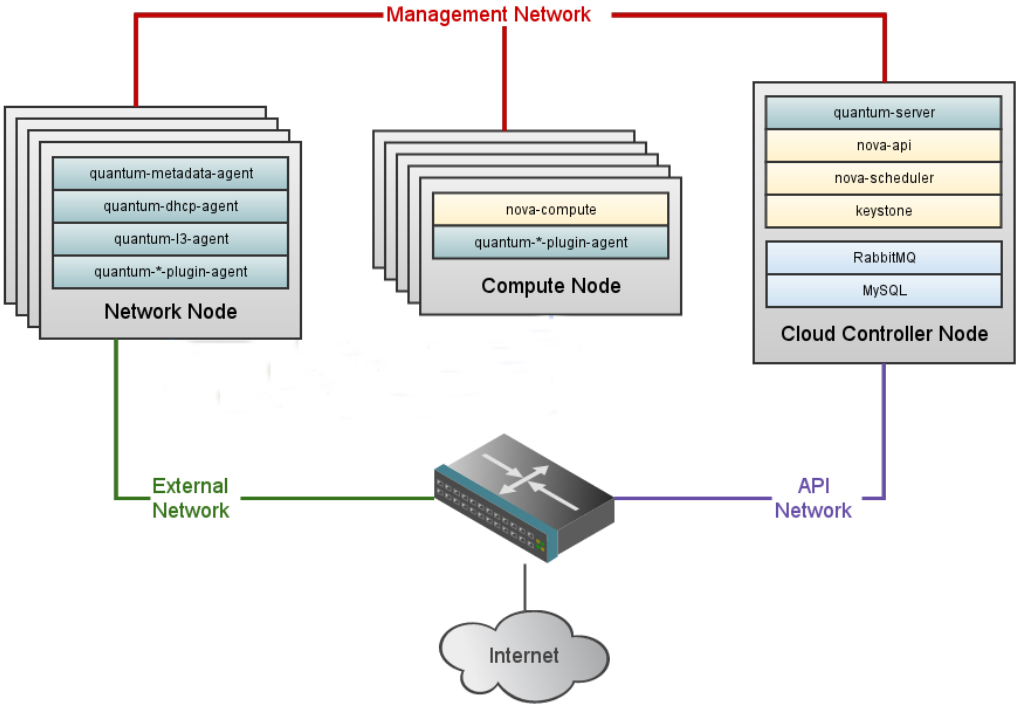


Figure 3.7. Nodes and Network for OpenStack
Source: `http://docs.openstack.org/grizzly/basic-install/apt/content/basic-install_architecture.html`

An example how the scaling works is given in Figure 3.8. SLAstic Lite uses Kieker to analyze the resource monitoring from the analysis nodes. First the components to execute commands on the command line and via SSH had been developed. Afterwards cloud specific controllers for start, shutdown and information gathering were implemented. These controllers use the ssh and command line components to execute the commands. In the next step strategies for the scaling had be developed. For the project an simple strategy had been developed, which starts an new instance if the load is higher then ninety percent and suspends an node if the load is lower then ten percent and shuts down the node if the load is lower then five percent. For distributing the workload, a simple load balancer was developed. This load balancer assigns the analysis nodes to the monitoring nodes. The load balancer is implemented as simple http service. To distribute the nodes the monitoring nodes are asking after a specified time interval for a new analysis node. For the distribution are several strategies possible, for the project a Round-Robin strategy was implemented, but also strategies which, for instance, take care of the load of an node

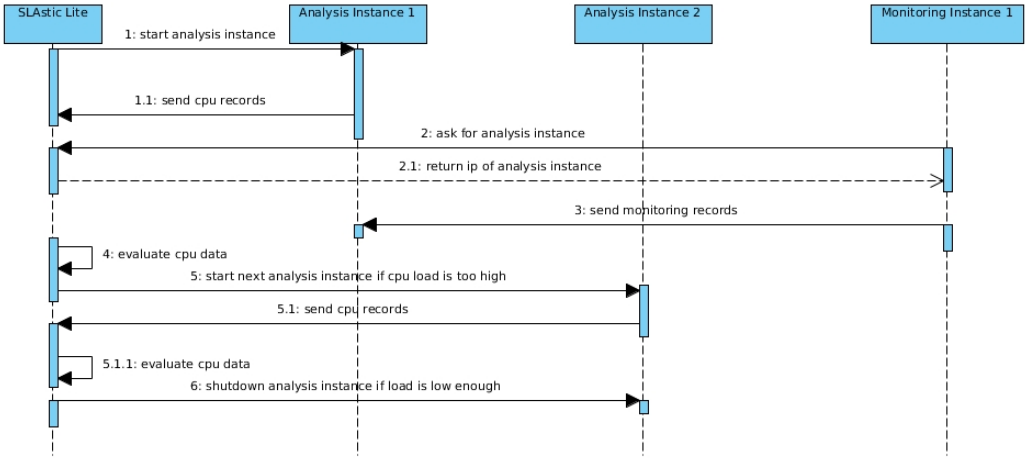


Figure 3.8. SLAstic Lite sequence diagram

are possible.

3.4 Evaluation

During the implementation, the evaluation of the new Kieker classes was conducted on the development computers. After this tests were successful kieker was tested in the cloud. For the evaluation of SLAstic Lite the test was started with component tests. First the components for ssh and command line execution have been tested. After this tests had been successful, the controller for controlling the cloud was tested separately. After the starting, stopping and information gathering was successful tested, an integration test with the missing components was started. As show on Figure 3.6 SLAstic Lite and the master node have been combined on a single machine, the cloud controller. For the workload generation a JBPM work flow has been used. These work flow had also been used to evaluate the new implemented filters.

Chapter 4

Communication Implementation in Kieker

For all technologies a writer and reader was implemented. The writers are implemented as asynchronous writers. Therefore, all writers implement the *AbstractAsyncWriter* and the *AbstractAsyncThread* of Kieker. The actual communication is done by the writer thread, the installation is shown in Figure 4.1. The writer, as Figure 4.2 shows, only reads all necessary properties from the given configuration and adds a writer thread to the monitoring controller of Kieker. The thread needs to implement a *constructor* and the two methods *consume(final IMonitoringRecord monitoringRecord)* and *cleanup()*. Both methods are of the return type *void*. In case of an error during consumption the *consume* method throws a exception. The *consume* method is responsible to send an *IMonitoringRecord* to an repository or direct to the analysis. The *cleanup* method is called on termination and is responsible that all used resources are closed and freed.
The readers implement the *AbstractReaderPlugin*. Therefore, the readers need to implement three methods and a *constructor*. In Figure 4.3 a class diagram on the example of the TCP-implementation is shown. The constructor reads like the writer all needed properties from the given configuration and sets an analysis controller. The first method is the *read* method which reads an record and passes it to the analysis. For this the *deliver* method of the super class can be used. As second method a *terminate* method must be implemented. This method is responsible for the clean termination of the reader and that all used resources are closed and freed.
The last method is the *getCurrentConfiguration*. This methods returns the current configuration.

4.1 Transmission Control Protocol

On the reader side a server-socket waits for a request to establish a connection. If the connection has successfully been established, the server-socket returns a *socket*. The data is read from the *socket* input stream by an *ObjectInputStream*. This is done in a loop while the reader is active. From the *ObjectInputStream* an object can be read directly by the *readObject* method. The read object can be delivered to the analysis without casting. The *setReuseAddress* flag is set. This will handle the problem of too many sockets in the TIME_-

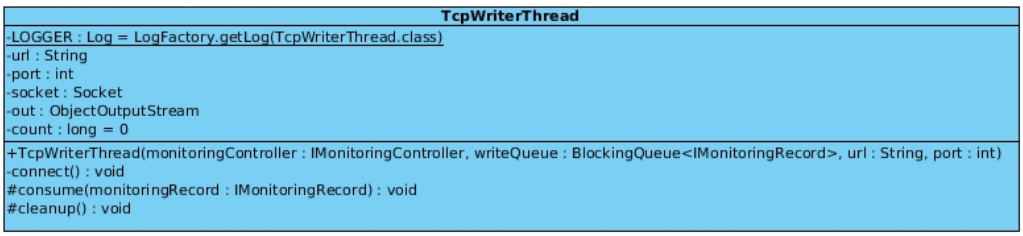


Figure 4.1. Class diagram of the writer thread on the example of TCP

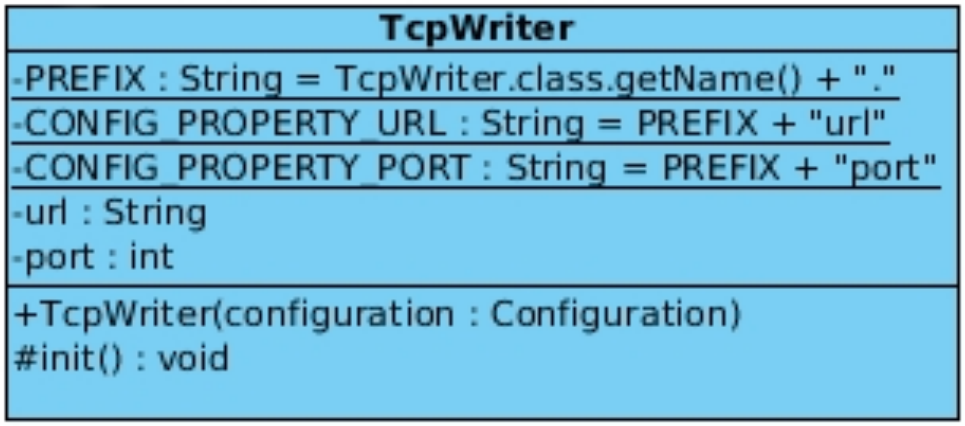


Figure 4.2. Class diagram of the writer on the example of TCP

WAIT state. Under normal conditions this should be no problem, it is only a problem if to many reconnects occur.
On the writer side a *socket* to the reader is opened and a *ObjectOutputStream* with the *socket* output stream. Objects can be send directly with the *writeObject* method. The *ObjectOutputStream* should be reseted after a few records because the *ObjectOutputStream* holds a reference to every send Object which will slow down the connection at first and will be followed by an out of memory exception. For this implementation the interval for the reset is set to three million records. As in the reader the *setReuseAddress* flag was set.

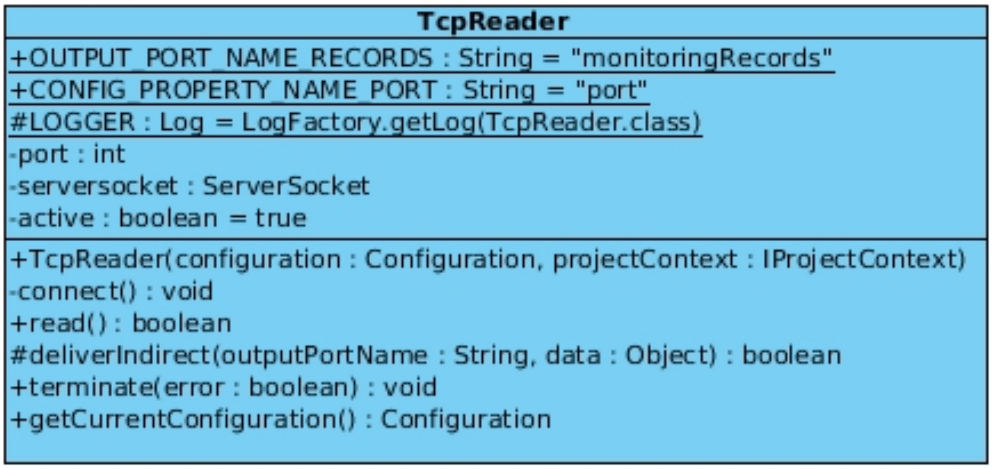


Figure 4.3. Class diagram of the reader on the example of TCP

4.2 User Datagram Protocol

UDP opens a datagram socket on the reader side and waits for incoming data. The data is sent as a byte array. UDP writes the incoming data into a local datagram packet, from which the byte array gets fetched. Afterwards the byte array is converted with streams to a object, which gets delivered to the analysis controller. The difference compared to TCP is that the data isn't send directly via an stream, it has to be extracted from the datagram packet.
The writer doesn't establish a connection. The writer packs the record with streams to an byte array. This byte array is packed into a datagram packet. The destination informations are added to the datagram packet and the packet gets send over the network.

4.3 Java Message Service on the Example of RabbitMQ

For RabbitMQ [*RabbitMQ*] a new library was added. This library contains the client to communicate with the RabbitMQ provider. RabbitMQ follows the point-to-point messaging concept of JMS. When the connection gets established, a factory with all needed informations like host and port is built and a connection is created. From the connection a channel needs to be created which declares the queue if not present at the provider. To receive messages a consumer is created which gets connected with the queue. After the connection is made, the consumer asks actively in a loop for the next message. The message arrives as byte array and will be passed through streams to an object. This object then gets delivered

to the analysis controller.
The writer establishes a connection in the same way the reader does. The record gets transformed with streams to a byte array. This byte array gets sent through the connection.

4.4 Java Remote Method Invocation

For RMI a new export class and interface was generated. This object provides a method for delivering the record. The reader exports this object via the RMI-Registry. After the export, the reader waits for the termination, the communication is done with the export object. An important aspect is that the reader needs to know his own address. Otherwise RMI uses the reverse lookup which will return an entry from /etc/hosts (on linux based operating systems).
The writer looks up the exported object in the RMI-Registry. After the writer has done the lookup, the records get sent as parameter with the method call on the remote object.
An important aspect is that all methods throw a RemoteException in the case of connection problems, which has to be handled.

Chapter 5

Performance Evaluation

5.1 GQM

Goal Question Metric (GQM) [Abib and Kirner 1999] is an goal oriented approach to specify a quality model for software engineering. It was developed by Prof. Victor R. Basili and Dr. David Weiss at the NASA Goddard Space Flight Center and is still used as a quality model there.
For GQM first an goal, which characterizes the project, the company, the mission statement and the measurement goals, has to be defined. For definition of such a goal, the question "Which goal should be reached with this measurements?" has to be answered. Afterwards questions have to be defined which characterize the goals more exactly. For this the question: "What should be measured?" has to be answered. In the last step a metric has to be defined which can answer the questions.
So GQM can be interpreted as a tree structure, which gets more and more specific from the root to the leafs. The gaol is the root, the questions are the nodes and the metrics are the leafs.

Goal

The goal of the evaluation is to find the best suited communication technology for the communication between the analysis and monitoring component of Kieker in the cloud.

Questions

The first question is: "Which technology provides the best performance for the communication?"
The second question is: "How reliable are the technologies?"

Metrics

As metric for the performance of the technologies the throughput of records per time interval has been defined.
For the reliability the loss rate of records in percent has been chosen.

5.2 Measurement Technique

To eliminate most external influences, it had been decided to build two jars for the measurement which include only the necessary parts. One which contains the writer and a controller which gives the ability to send *OperationExecutionRecords* over an specified period of time. When the time limit is exceeded, the number of send records is printed to the command line. On startup the writer and all the parameters needed by the writer have to be specified via the command line. Here also the time interval in which records should be sent has to be specified.
The second jar contains the reader and a modified *CountingThroughputFilter*. The filter counts the number of received records and the number of received records per time interval. As in the jar with the writer, the reader and all his parameters have to be specified via command line at the startup. When the execution of the jar gets terminated, the number of received records and the number of received records for every time interval is printed to the command line.

5.3 Experimental Setup

In the following, the hardware configuration, the deployment and the measurement technique are presented.

5.3.1 Hardware Configuration

For the evaluation two servers with Ubuntu 12.04 LTS [*Ubuntu*] have been used. The server for receiving the data had a dual core processor with 2.5 GHz per core and 4GB of RAM. This server also contained the RabbitMQ server in the version 3.1.5.
The server for sending the data had an Intel core i3 processor with 2.93 GHz per core and 8GB RAM. The servers have been connected through a 100 Mbit network switch with five meter cables each.

5.3.2 Deployment

The jars have been placed on the two different servers. First the jar with the reader needed to be started for every experiment and afterwards the jar with the writer. After an experiment finished the data printed to the command line had to be saved and the next experiment could be started. For the evaluation a interval size of one minute had been used. Every technology was evaluated in five experiments. Four with a length of ten minutes and two seconds and one with a length of twenty minutes and two seconds. The two seconds were added to guarantee that on the reader side also for the last minute a throughput could be determined.

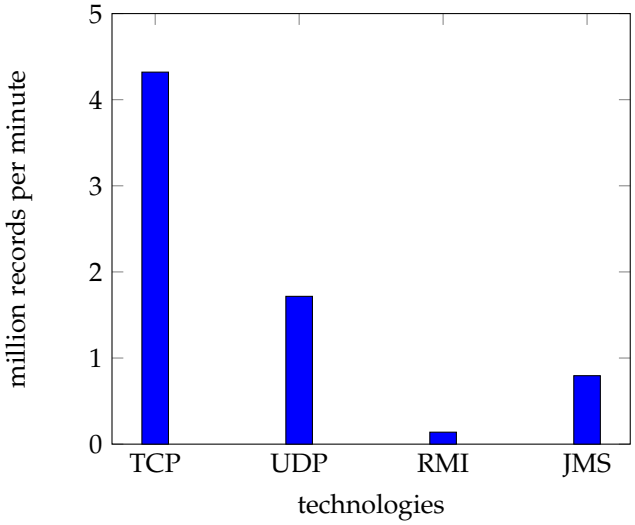


Figure 5.1. Average throughput of the five experiments

5.4 Results

In the evaluation the fastest technology was TCP, followed by UDP then JMS and the slowest technology is RMI. The results are visualized in Figure 5.1 TCP has an average throughput as shown in Table 5.1 of 4.3 million records per minute. This is 2.5 times faster then UDP, which can process 1.7 million records per minute. UDP is 2.1 times faster then JMS which has an average throughput of nearly eight hundred thousand records per minute, what is 5.7 times faster then RMI is, which can only process one hundred forty thousand records per minute. The throughput for every technology for every experiment is shown in the appendix.
The highest standard deviation as shown in Table 5.3 with 138,579.01 has TCP, again followed by UDP with 5,413.91 and JMS with 3,527.34. The lowest standard deviation has RMI with 309.61.
A loss rate is only present with UDP. The average loss rate, displayed in Table 5.2, of UDP is six percent.

5.5 Discussion of the Results

We expected that UDP is faster then TCP, because UDP is a connection less protocol and doesn't inherits the overhead for the connection and the acknowledgments like TCP does. As explanation two possibilities have been found. On the one hand, for UDP for every record a new byte array and datagram needs to be packed. First the record gets packed

Experiment	TCP	UDP	RMI	JMS
1	4,413,257.80	1,719,275.00	138,870.00	792,831.70
2	4,452,472.00	1,708,467.80	138,785.90	792,834.30
3	4,089,780.40	1,724,217.80	138,767.00	794,132.60
4	4,414,080.60	1,717,469.10	138,873.80	793,327.50
5	4,230,990.45	1,712,778.65	139,590.65	802,019.50
average	4,320,116.25	1,716,442.27	138,977.47	795,029.12

Table 5.1. Average throughput per minute in records

Technology	1	2	3	4	5	average
TCP	0.00	0.00	0.00	0.00	0.00	0.00
UDP	0.06	0.07	0.06	0.06	0.06	0.06
RMI	0.00	0.00	0.00	0.00	0.00	0.00
JMS	0.00	0.00	0.00	0.00	0.00	0.00

Table 5.2. Loss rate per experiment in percent

by a stream into a byte array. This byte array is packed into the datagram packet which gets sent through the network. TCP uses the stream from the connection and can send the records directly. On the other hand, TCP use an algorithm called nagle's algorithm [Minshall et al. 2000], this algorithm is used by default with TCP in Java. The algorithm improves the performance of TCP by reducing the number of packets, which need to be sent over the network. The improvement is that the number of small packets to sent gets reduced. For this the data is buffered until a minimum of segments to send is achieved. This algorithm isn't applicable for UDP because the data can't be buffered before sent. The buffering can also be the explanation for the high standard deviation by TCP.
That JMS is slower than TCP or UDP had been expected. The message\record has to be send twice. First from the message producer, the writer, to the JMS provider and then from the JMS provider to the message consumer, which is the reader. In the meantime the record has to be stored to the queue or topic and read again, what also takes time.

Technology	standard deviation
TCP	138,579.01
UDP	5,413.91
RMI	309.61
JMS	3,527.34

Table 5.3. Standard deviation of the throughput

RMI uses TCP as basic transport protocol, which indicates that it can't be faster than TCP. On top of this a overhead has to be managed, for example, for the return value. So a bigger amount of data has to be exchanged and more calls are made, what slows down the communication.
The evaluation also showed that with increasing throughput of an technology the standard deviation increases. This is reasonable because with higher throughput the impact of small interferences is higher then with a smaller throughput.
Only UDP did not provide reliability for receiving sent messages. This circumstance stems from the fact that UDP is an connection less protocol and the data only gets packed and sent without any acknowledgment. The missing acknowledgment makes it impossible to react on damaged or lost packets like TCP does.

5.6 Threats to Validity

Implementation

The implementation was done with care, but it is still possible that during the implementation mistakes had been made. For example, it is possible that important flags have been forgotten, which could be important for the performance of a technology. It is also possible that better techniques are available which haven't been know and so have been ignored.

Server

The systems for the evaluation haven't been optimized. So interferences by the system like checking for new updates are possible. A second thing about the server is that they had been normal computer hardware and no server hardware.

Network

The 100 Mbit network may have reduced the technologies in their full performance. So maybe all technologies can be more effective with a 1 Gbit network.
It is also possible that breaks in the network cables prevented an optimal performance.

Chapter 6

Related Work

In the past different evaluations of the performance of communication technologies have been conducted. Most of them had the goal to evaluate low level technologies, normally TCP or UDP, in a special environment like in a wireless LAN [Xylomenos and Polyzos 1999] or on mobile networks [Lee et al. 2001]. Other works tried to optimize special technologies like TCP. These works had, for example, the goal to answer the question whether to use the nagle algorithm or not [Minshall et al. 2000].

Chapter 7

Conclusions and Outlook

7.1 Summary

The goal of this thesis was to evaluate the best suited communication technology for the monitoring software Kieker. To achieve this, four technologies were successfully implemented for use in a productive environment. The chosen technologies were TCP, UDP, RMI, and JMS on the example of RabbitMQ. The four technologies have been evaluated with the defined metrics throughput per minute and loss rate. Only UDP did have an loss rate from six percent. All other technologies are reliable. In our evaluation TCP provides the best throughput accompanying no loss rate. TCP is followed by UDP and JMS. The slowest technology is RMI.

7.2 Discussion

The evaluation done by this thesis showed that for a good performance the use of low level technologies like TCP or UDP is fundamental. High level technologies like RMI or JMS normally use low level technologies for the communication and have to deal with a overhead on top of this. The evaluation also showed that the use of UDP has to be deliberated. A loss rate of six percent is too high for most applications.
The evaluation only toke care of small data packets. This has to be remembered when working with bigger data packets.

7.3 Future Work

Optimizations

As mentioned in Section 5.6 the implementation is done with care, but still there needs to be evaluated which flags can be helpful to improve the performance of the implemented technologies. Another thing which has to be optimized is the reset of the *ObjectOutputStream* in the TCP writer implementation. At the moment this reset is done after three million records, but this doesn't have to be the optimum. Furthermore, it should be evaluated if the loss rate of UDP can be optimized.

Message Batching

Another important issue which is left open by this thesis, is how effective message batching would be. The discussion of the results showed that UDP, for example, is slow for many small packets. This indicates that message batching could significantly increase the performance of UDP. For message batching, it should be evaluated how many records have to batched before sending, for the optimum performance.

Multi Threading

In the current implementation, writing and reading is done with a single thread. It would be definitely interesting how the communication can be optimized by multi threading. At the moment this would be time-consuming, because some major changes in the Kieker architecture are necessary to do this.

Other Technologies

Furthermore, it should be evaluated if there are other more effective technologies than the ones used in this thesis evaluated technologies. This thesis only evaluated four technologies out of many. Maybe JMX or other JMS provider then RabbitMQ like ActiveMQ [*ActiveMQ*] are more effective.

Bibliography

[Abib and Kirner 1999] J. C. Abib and T. G. Kirner. A GQM-based tool to support the development of software quality measurement plans. *SIGSOFT Softw. Eng. Notes* 24.4 (July 1999), pages 75–80. (Cited on page 23)

[*ActiveMQ*] ActiveMQ. last visited: 2013-09-23. URL: http://activemq.apache.org/. (Cited on pages 7 and 32)

[*Amazon EC2*] Amazon EC2. last visited: 2013-06-14. URL: http://aws.amazon.com/ec2/. (Cited on page 1)

[AspectJ] AspectJ language extension. Eclipse Foundation. URL: http://www.eclipse.org/aspectj/. (Cited on page 3)

[Blackburn et al. 2006] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis. In: *Proceedings of the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 2006)*. ACM, 2006, pages 169–190. (Cited on page 6)

[Cerf et al. 1974] Cerf, Dalal, and Sunshine. RFC 675. last visited: 2013-06-14. 1974. URL: http://www.rfc-editor.org/rfc/rfc675.txt. (Cited on page 6)

[Defense Advanced Research Projects Agency 1991] Defense Advanced Research Projects Agency. RFC 791. last visited: 2013-06-18. Sept. 1991. URL: http://www.rfc-editor.org/rfc/rfc791.txt. (Cited on page 7)

[*Eucalyptus*] Eucalyptus. last visited: 2013-06-21. URL: http://www.eucalyptus.com/. (Cited on page 1)

[Fittkau 2012] F. Fittkau. Online trace visualization for system and program comprehension in large software landscapes. In: *KoSSE-Symposium Application Performance Management (Kieker Days 2012)*. 2012. (Cited on page 1)

[Fittkau et al. 2013] F. Fittkau, J. Waller, C. Wulf, and W. Hasselbring. Live trace visualization for comprehending large software landscapes: The ExplorViz approach. In: *1st IEEE International Working Conference on Software Visualization (VISSOFT 2013)*. 2013. (Cited on page 4)

[Georges et al. 2007] A. Georges, D. Buytaert, and L. Eeckhout. Statistically rigorous Java performance evaluation. In: *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications (OOPSLA 2007)*. ACM, 2007, pages 57–76. (Cited on page 2)

[Hapner et al. 2002] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout. Java Message Service. last visited: 2013-06-14. 2002. URL: http://download.oracle.com/otn-pub/jcp/7195-jms-1.1-fr-spec-oth-JSpec/jms-1_1-fr-spec.pdf. (Cited on pages 1 and 7)

[*iCloud*] iCloud. last visited: 2013-09-24. URL: https://www.icloud.com/. (Cited on page 1)

[ITU 1994] T. S. S. O. ITU. last visited: 2013-06-18. July 1994. URL: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.200-199407-I!!PDF-E&type=items. (Cited on page 7)

[*Java Remote Method Invocation*] Java Remote Method Invocation. last visited: 2013-06-24. URL: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html. (Cited on page 8)

[*JBPM*] JBPM. last visited: 2013-09-05. URL: http://www.jboss.org/jbpm/. (Cited on page 13)

[Basic JMS API Concepts] Java Message Service. last visited: 2013-05-26. Oracle. URL: http://docs.oracle.com/javaee/6/tutorial/doc/bncdx.html. (Cited on page 7)

[Kalibera and Jones 2013] T. Kalibera and R. Jones. Rigorous benchmarking in reasonable time. In: *Proceedings of the ACM SIGPLAN International Symposium on Memory Management (ISMM '13)*. ACM, June 2013, pages 2–4. (Cited on page 6)

[Lee et al. 2001] S. B. Lee, G. S. Ahn, and A. T. Campbell. Improving udp and tcp performance in mobile ad hoc networks with insignia. *Comm. Mag.* 39.6 (June 2001), pages 156–165. (Cited on page 29)

[Minshall et al. 2000] G. Minshall, Y. Saito, J. C. Mogul, and B. Verghese. Application performance pitfalls and tcp's nagle algorithm. *SIGMETRICS Perform. Eval. Rev.* 27.4 (Mar. 2000), pages 36–44. (Cited on pages 26 and 29)

[*OpenStack*] OpenStack. last visited: 2013-06-21. URL: http://www.openstack.org/. (Cited on page 1)

[*ownCloud*] ownCloud. last visited: 2013-09-24. URL: http://owncloud.org/. (Cited on page 1)

[Process 2006] J. C. Process. Java Mamagment Extension. last visited: 2013-06-14. 2006. URL: http://download.oracle.com/otn-pub/jcp/jmx-1.4-mrel3-jsr003a-oth-JSpec/jmx-1_4-mrel3-spec.pdf. (Cited on page 1)

[*Pubflow*] Pubflow. last visited: 2013-09-05. URL: http://www.pubflow.uni-kiel.de/. (Cited on page 13)

[*RabbitMQ*] RabbitMQ. last visited: 2013-09-02. Pivotal. URL: http://www.rabbitmq.com/. (Cited on pages 7 and 21)

[RFC 1980] RFC. RFC 768. last visited: 2013-06-14. 1980. URL: http://www.rfc-editor.org/rfc/rfc768.txt. (Cited on page 7)

[Rohr et al. 2008] M. Rohr, A. van Hoorn, J. Matevska, N. Sommer, L. Stoever, S. Giesecke, and W. Hasselbring. Kieker: continuous monitoring and on demand visualization of java software behavior. In: *Proceedings of the IASTED International Conference on Software Engineering 2008 (SE'08)*. Edited by C. Pahl. Anaheim, CA, USA: ACTA Press, 2008, pages 80–85. (Cited on page 11)

[*Ubuntu*] Ubuntu. last visited: 2013-09-02. Canonical Ltd. URL: http://www.ubuntu.com/. (Cited on page 24)

[Van Hoorn et al. 2009] A. van Hoorn, M. Rohr, A. Gul, and W. Hasselbring. An adaptation framework enabling resource-efficient operation of software systems. In: *Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010*. WUP '09. Cape Town, South Africa: ACM, 2009, pages 41–44. (Cited on page 13)

[Van Hoorn et al. 2012] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In: *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*. Boston, Massachusetts, USA, April 22–25, 2012: ACM, Apr. 2012, pages 247–248. (Cited on pages 1 and 3)

[Xylomenos and Polyzos 1999] G. Xylomenos and G. Polyzos. Tcp and udp performance over a wireless lan. In: *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Volume 2. 1999, 439–446 vol.2. (Cited on page 29)

Appendix A

Acronyms

AQMP Advanced Message Queuing Protocol.

GQM Goal Question Metric.

JMS Java Message Service.

RMI Remote Method Invocation.

TCP Transmisson Control Protocol.

UDP User Datagram Protocol.

UML Unified Modeling Language.

Appendix B

Detailed Results for the Experiments

Minutes	TCP	UDP	RMI	JMS
1	4,207,289	1,700,260	133,616	745,483
2	4,447,404	1,727,082	139,766	801,700
3	4,450,471	1,729,440	139,627	797,838
4	4,343,763	1,724,501	139,823	791,718
5	4,509,590	1,723,209	139,388	796,844
6	4,330,236	1,716,275	139,483	796,879
7	4,454,591	1,716,222	139,248	801,843
8	4,419,095	1,719,814	139,522	795,566
9	4,497,599	1,723,456	139,219	798,891
10	4,472,540	1,712,491	139,008	801,555
total	44,132,578	17,192,750	1,388,700	7,928,317

Table B.1. Results for the first experiment in records per minute

Minutes	TCP	UDP	RMI	JMS
1	4,347,232	1,689,491	133,352	750,948
2	4,433,250	1,712,463	139,713	792,553
3	4,485,504	1,711,806	139,580	794,591
4	4,372,105	1,713,308	139,416	800,237
5	4,547,228	1,706,251	139,178	791,226
6	4,437,999	1,708,628	139,352	800,254
7	4,401,511	1,710,499	139,382	802,020
8	4,448,082	1,713,524	139,449	797,514
9	4,496,950	1,707,414	139,260	798,696
10	4,554,859	1,711,294	139,177	800,304
total	44,524,720	17,084,678	1,387,859	7,928,343

Table B.2. Results for the second experiment in records per minute

Minutes	TCP	UDP	RMI	JMS
1	3,915,251	1,687,825	133,024	746,898
2	4,092,212	1,723,851	139,584	797,139
3	3,956,530	1,728,849	139,563	798,053
4	4,120,412	1,725,764	139,747	798,718
5	4,054,245	1,730,426	139,288	801,622
6	4,144,429	1,722,800	139,213	805,555
7	4,045,442	1,731,578	139,396	797,257
8	4,104,646	1,729,230	139,297	800,896
9	4,261,220	1,728,062	139,295	795,118
10	4,203,417	1,733,793	139,263	800,070
total	40,897,804	17,242,178	1,387,670	7,941,326

Table B.3. Results for the third experiment in records per minute

Minutes	TCP	UDP	RMI	JMS
1	4,301,107	1,696,995	133,487	745,127
2	4,359,183	1,726,598	139,693	793,657
3	4,403,059	1,721,714	139,590	801,599
4	4,137,664	1,727,730	139,607	801,602
5	4,522,281	1,718,563	138,830	806,102
6	4,448,979	1,718,186	139,495	791,803
7	4,420,857	1,719,200	139,508	795,910
8	4,439,725	1,710,720	139,616	795,048
9	4,534,073	1,718,941	139,628	805,895
10	4,573,878	1,716,044	139,284	796,532
total	44,140,806	17,174,691	1,388,738	7,933,275

Table B.4. Results for the fourth experiment in records per minute

Minutes	TCP	UDP	RMI	JMS
1	3,976,330	1,687,184	133,657	749,920
2	4,143,084	1,713,612	141,162	807,277
3	3,899,550	1,707,119	140,129	809,327
4	4,158,180	1,712,460	139,694	807,699
5	4,279,828	1,714,158	139,306	810,918
6	4,146,877	1,716,299	139,679	800,316
7	4,256,806	1,713,767	142,314	798,139
8	4,216,575	1,712,176	140,216	801,969
9	4,354,247	1,715,793	139,286	805,360
10	4,259,475	1,718,955	139,217	809,046
11	4,214,058	1,715,072	139,215	802,692
12	4,251,210	1,712,013	139,428	799,882
13	4,383,320	1,711,172	139,177	797,203
14	4,241,619	1,713,196	139,155	805,097
15	4,217,469	1,716,373	139,255	810,288
16	4,391,897	1,716,189	140,679	807,821
17	4,285,343	1,714,604	140,688	811,050
18	4,305,738	1,712,199	140,989	802,287
19	4,247,531	1,717,791	139,450	803,843
20	4,390,672	1,715,441	139,117	800,256
total	84,619,809	34,255,573	2,791,813	16,040,390

Table B.5. Results for the fifth experiment in records per minute

Experiment	UDP	RMI	JMS
1	2.5669295488	31.7797782098	5.566449727
2	2.6061199397	32.0815875388	5.6158922489
3	2.3719627532	29.4722837562	5.1499968645
4	2.5701077242	31.7848334243	5.5640080547
5	2.4702494102	30.3099845871	5.2754209218
average	2.5170738752	31.0856935032	5.4343535634

Table B.6. TCP compared to UDP, RMI and JMS

Experiment	RMI	JMS
1	12.3804637431	2.1685245431
2	12.3100963426	2.1548863363
3	12.4252725792	2.1711963468
4	12.367121084	2.1648929351
5	12.2700098466	2.1355823019
average	12.3505927191	2.1590164926

Table B.7. UDP compared to RMI and JMS

Experiment	RMI
1	5.7091646864
2	5.712642999
3	5.7227770291
4	5.7125786145
5	5.7455101756
average	5.7205347009

Table B.8. JMS compared to RMI

Appendix C

Attachment

Attached to this work is an CD which contains the source code, developed during this work, this document as PDF and the compiled jar files. The structure of the CD is explained by an README.txt on the CD.