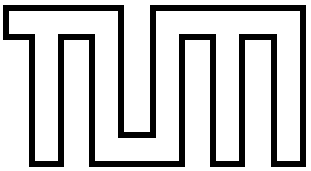


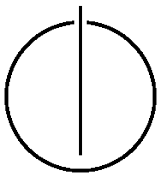
FAKULTÄT FÜR INFORMATIK

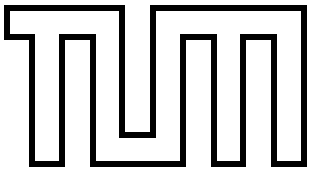
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Supporting Capacity Planning of Cloud Computing Data Centers with Long Term Trend Analysis of Performance Monitoring Data

Markus Fensterer





FAKULTÄT FÜR INFORMATIK

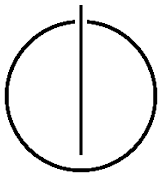
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Supporting Capacity Planning of Cloud Computing Data Centers with Long Term Trend Analysis of Performance Monitoring Data

Unterstützung der Kapazitätsplanung in Cloud Computing Rechenzentren durch Langzeit-Trendanalysen mit Leistungsdaten

Author:	Markus Fensterer
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	M. Sc. Matheus Hauder
Date:	Juli 16, 2012



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this bachelor's thesis only supported by declared resources.

München, den 16. Juli 2012 Markus Fensterer

Abstract

The aim of this thesis is the development of a forecasting tool for resource utilization metrics, as the CPU utilization, in order to support capacity planners of Infrastructure-as-a-Service (IaaS) providers in their decisions. The whole process of measuring, collecting and storing of performance monitoring data is described, as well as the statistical forecasting and the detection of threshold violations. The main focus of the thesis lies on the statistical modeling, forecasting and visualization of the data. The forecasting functionality of the system allows capacity planners a look into the future and in combination with previous performance data long term trends can be visualized. Autoregressive models are used for forecasting.

The data model of the Finanz Informatik Technologie Service GmbH (FI-TS) for storing performance monitoring data is evaluated and its maturity is shown. An introduction to the noSQL database Cassandra which FI-TS uses for storing monitoring data is given. Furthermore, a navigation concept based on mockups is described which simplifies the navigation through the performance data. In order to allow users to drill down to the cause of threshold violations an overview chart for the threshold violations is introduced.

The implementation comprises the development of a web dashboard that visualizes the monitoring data as well as the forecasts and threshold violations. Measuring, collecting and persisting the monitoring data are not part of the implementation because these parts already exist at the FI-TS. Nevertheless, the system was implemented to a large extend independently from a specific IaaS provider. For evaluation purposes, the system was deployed to the infrastructure of FI-TS to gain insights into the system's prediction accuracy. With a small dataset it is shown that an forecast accuracy in terms of the mean percentage error (MPE) of 24.5% is reached; i.e.: the predictions differed on average by 24.5% from the actually observed data.

Zusammenfassung

Ziel dieser Arbeit ist die Entwicklung eines Tools zur Vorhersage von Leistungswerten, wie der CPU Last, um Kapazitätsplaner von IaaS Anbietern in ihren Entscheidungen zu unterstützen. Der gesamte Prozess von der Messung, Sammlung und Speicherung der Leistungsdaten, über die statistische Vorhersage, bis hin zur Erkennung von Grenzwertüberschreitungen und der Visualisierung der Daten wird beschrieben. Mit der Vorhersage-Funktionalität des Systems können zukünftige Entwicklungen abgeschätzt werden und in Verbindung mit der Darstellung vorausgegangener Messungen werden Langzeit-Trends für die Nutzer sichtbar. Für die Vorhersagen werden dabei autoregressive Modelle verwendet.

Das Datenmodell der Finanz Informatik Technologie Service GmbH (FI-TS) zur Speicherung von Leistungsdaten wird evaluiert und als ausgereift erachtet. Eine Einführung in die noSQL Datenbank Cassandra, die FI-TS zur Speicherung der Kapazitätsdaten verwendet, wird im Rahmen dieser Arbeit gegeben. Weiterhin beschreibt die Arbeit ein Navigationskonzept auf Basis von Mockups, welches das Navigieren durch die Leistungsdaten vereinfacht. Eine Übersichtsdarstellung für Grenzwertüberschreitungen wird eingeführt, die es Benutzern ermöglicht einfach zur Ursache von den Überschreitungen zu gelangen.

Die Implementierung besteht aus einem Web Dashbhoard, das die Leistungsdaten, die Vorhersagen und die Grenzwertüberschreitungen darstellt. Die Komponenten zur Messung, Sammlung und Speicherung von Leistungsdaten sind nicht Teil der Implementierung, da diese Komponenten bereits bei der FI-TS bestehen. Das entwickelte System ist aber größtenteils unabhängig von einem speziellen IaaS Anbieter. Zur Bewertung des Systems wurde es in der IaaS Umgebung der FI-TS eingerichtet. Die Vorhersagegenauigkeit des Systems innerhalb eines Testdatensatzes beträgt 24.5%, wenn der “mean percentage error” (MPE) als Maß verwendet wird. Das heißt: die Vorhersagen lagen in diesem Datensatz im Mittel 24.5% von den tatsächlich gemessenen Werten entfernt.

Contents

List of Figures

List of Tables

Part I.

Introduction

1. Motivation and problem description

Cloud computing is one of the most rapidly evolving markets in the IT sector. Morgan Stanley analyzed the cloud computing ecosystem in the blue paper "Cloud Computing Takes Off" [20]. It states that the Infrastructure-as-a-Service (IaaS) section of the cloud computing market will experience an annual growth rate of about 60% from 2011 until 2013. The biggest players within the IaaS market are Amazon and Rackspace, but there also exist smaller companies like Joyent, GoGrid, OpSource and Linode which also experience high growth rates [20].

Amazon already started their IaaS offering called EC2 in 2006. Now, six years later, Google introduced also an IaaS system, called Google Compute Engine (GCE)[1]. The development of GCE indicates the continuing relevance of this topic for the industry.

But what is cloud computing really about? The National Institute of Standards and Technology defines cloud computing as "[...] a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [21].

IaaS is one of the three common service models of cloud computing (besides Platform-as-a-Service - PaaS and Software-as-a-Service - SaaS). With IaaS, customers are able to rent computing resources like CPUs, memory and persistent storage on a pay-per-use basis. Users are provied with full access to the operating system of the servers [21].

With the raise of cloud computing, more and more traditional data center operators are on the way to become an "Infrastructure as a Service" (IaaS) provider because the idea of IaaS promises cost-savings through server consolidation as well as more flexibility for the data center operators themselves and their customers. Before the IaaS era, the contract period for renting or managing servers was commonly set to several years. The motivation for many customers for moving to IaaS emerges from the ability to gain a lower time-to-market for their products and to be more flexible in decisions about their hardware landscape. Also a survey found, that 50% of the budget of IT projects are consumed by costs for infrastructure and maintenance. IaaS aims to lower both of these cost units [20]. Usually, IaaS contracts do not contain a fixed set of servers which the data center has to provide because customers should be able to start and stop servers at anytime. This implies that the demand of servers for an IaaS provider is frequently fluctuating and capacity planners have to react fast to these fluctuations in order to offer a good service quality to their customers. Thus, capacity planners have to understand these demand fluctuations as well as possible. Otherwise they will faciliate either too much or to little hardware resources ending up in a degraded service quality for the customers: either the costs for

[1] http://cloud.google.com/products/compute-engine.html, last accessed on July 9, 2012

the customers increase because the costs of operation increase for the IaaS provider or customers will experience that they cannot start new virtual machines anymore if the number of servers is too small.

The questions that come up to capacity planners of an IaaS provider in terms of hardware and software can be summarized as follows:

1. When should new servers be ordered?
2. Are there any software bottlenecks existing in any system of the enterprise?
3. When will the data center need how many servers?
4. When should a server be shutdown?

This thesis pesents a system that supports capacity planners of IaaS providers in these questions. The system is based on measurements of performance metrics of servers and uses statistical forecasting on this data to predict future resource utilizations. Thereby, a noSQL database is used for storing the performance data.

2. Outline

This thesis continues with the definition of the scope in chapter 3 by summarizing the covered topics and differentiating from already existing components at the Finanz Informatik Technologie Service GmbH (FI-TS). Related work to the topic of capacity planning as well as commercial systems for capacity planning are described in chapter 4. In the following, the role of capacity planning is explained from a business perspective (chapter 5). An overview over the company FI-TS and its services is given in chapter 6. Because the current system at FI-TS relies on the noSQL database Cassandra, this technology is explained in chapter 7. The current data model of FI-TS for storing monitoring data is evaluated in chapter 8. Subsequently, statistical methods that are used by the software component are explained in chapter 9, with a focus on Autoregression (section 9.2). The different activities of a software engineering project are examined in separate chapters: requirements engineering (chapter 10), system design (chapter 11) and implementation (chapter 12). At the end, an evaluation of the constructed system is given, possible extensions of the system are proposed (chapter 14) and a conclusion about the whole thesis is given (chapter 15).

3. Scope

There exist two approaches to understand the hardware demand fluctuations that are caused by the IaaS concept: firstly, analyzing the fluctuations of virtual servers needed and secondly, analyzing the evolution of performance metrics, like CPU utilization, of servers. In this work the latter approach will be followed and combined with an automated performance analysis which uses statistical forecasting.

This thesis also covers an implementation part that was carried out in the manner of a software engineering project. Goal of the implementation is a web dashboard that supports capacity planners of IaaS providers in their decisions about the questions posed in chapter 1 with automated forecasting on performance monitoring data. The system is implemented as an extension to an existing management dashboard that runs on the intranet of FI-TS. The implementation uses code components that were developed by FI-TS, including measuring, collecting and persisting the performance monitoring data gathered from their servers. The part of the system that was developed during the thesis accesses this data, builds statistical models for it, calculates forecasts by using this models and detects threshold violations within the data. Besides the visualization of monitoring data, also the forecasted values and the threshold violations are visualized. An overview of the scope of the thesis and the course of actions the implementation comprises is shown in figure 3.1.

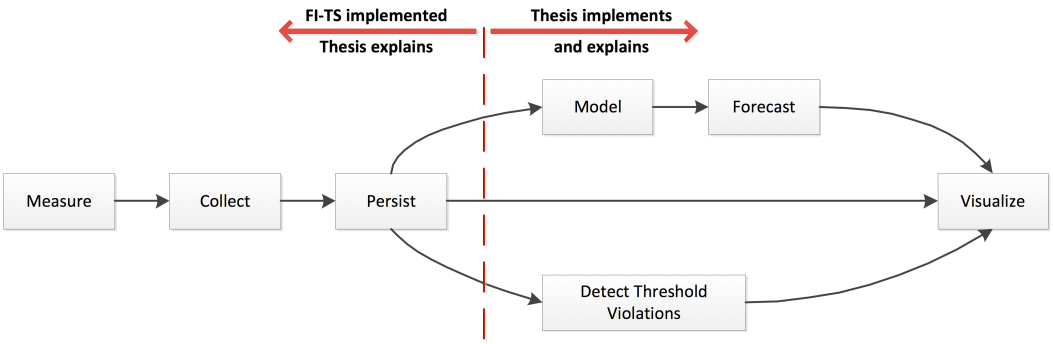


Figure 3.1: Scope of the thesis and the course of actions the system comprises

Autoregressive models were chosen as statistical method to estimate the forecasts. This was already defined at the beginning of the thesis because this method is widely spread in related works (see chapter 4) and good results have been gained within a prototypical analysis of a small data sample. Furthermore, an evaluation of the performance of different models would go beyond the scope of this thesis.

4. Related Work

4.1. Capacity planning

Wolski et al. ([25], [26]) developed a system called "Network Weather Service" (NWS) which is intended to make short-term resource utilization predictions in grid computing environments. He describes the different forecasting methods used and the framework behind the service. The idea of the system is to use several modeling approaches (mean based, median based, autoregressive models) in parallel and dynamically selecting the most promising approach using accuracy measures (e.g. the mean squared error or mean percentage error). The levels of aggregation are restricted to one minute, five minutes and 15 minutes. Higher aggregation levels are not supported. Furthermore the aggregation of data is done on the server where the performance data is measured and not on the storage side, as within the system of this thesis. The chart generation is done on the server side which restricts the possibilities of users to interact with it. In contrast to that the system of this thesis generates the charts on the client side. Furthermore, NWS has no component for detecting threshold violations within the performance data and it does not consider navigational issues for users to find their way through all the metrics that are measured.

Dinda [7] proposed a system called RPS which is similar to NWS. RPS estimates forecasts for CPU loads and network bandwith utilization in distributed systems. In contrast to NWS and to the system of this thesis, it includes a wide range of models to use for forecasting: linear as well as nonlinear models (e.g. wavelet analysis). Dinda made good experiences with autoregressive models of order 16, to predict the CPU utilization using 600 datapoints. The distinguishing factors between RPS and the system developed in this thesis are the same as the differences to NWS.

Prevost et al. [18] presented a framework for the optimization of cloud resource allocations by analyzing and forecasting performance data like the network load. It was shown that the performance of Autoregression outranged neural networks. In contrast to this thesis, this framework covers just the modeling part, but not an implementation part.

Another modeling approach was proposed by Liang et al. [17]. They indroduced a multi-resource prediction model that incorporates correlations between different time series of performance data. It was shown that their approach significantly reduces prediction errors in comparison to traditional approaches which use autoregressive models. However, this work does not cover an implementation part.

Research around frequency based analysis of performance measurements was done by Daniel Gmach et al. in several papers ([11], [10]). But these papers are not concerned with usability questions.

4.2. Commercial products

There also exist some commercial products that are concerned about supporting capacity planners.

HP OpenView Performance Manager[1] is the product that is closest to the system developed in this thesis, but it has far more features. It includes real-time graphing of performance metrics, several graphing options for smoothing curves, an integrated support for generating reports about performance metrics and a notification service for alarms that emerge. These features are not part of the system of this thesis.

CA Capacity Manager[2] aims to help capacity planners to consolidate workloads and thus, gaining a higher resource utilization. Therefore, it displays performance measurements of servers and estimates forecasts using linear regression. In contrast to this thesis, the CA Capacity Manager also considers correlations between the different performance metrics and it makes concrete proposals for consolidation decisions. However, it does not use advanced modeling techniques like Autoregression and the UI is not realized as a web interface but as a desktop application.

VMware Capacity Planner[3] is a web based tool which analyzes and models performance metrics. It is also able to model what-if scenarios i.e., what would happen to the performance measurements if a server instance is migrated to an other server. Such a scenario modeling feature is not part of the thesis.

[1] https://h20392.www2.hp.com/portal/swdepot/displayProductInfo.do?productNumber=PERFMINFO, last accessed on July 9, 2012

[2] http://www.ca.com/de/capacity-manager.aspx, last accessed on July 9, 2012

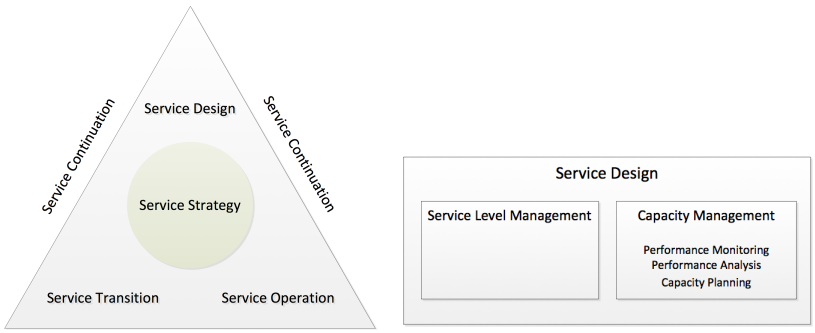
[3] http://www.vmware.com/files/de/pdf/datasheet_capacity_planner_de.pdf, last accessed on July 9, 2012

Part II.

Overview

5. Role of capacity planning within ITIL

In this chapter the significance of capacity planning will be described from a business perspective using the framework provided by the IT Infrastructure Library (ITIL). The first version of ITIL was developed back in 1989 by order of a british govermental institution. Its aim was right form the start on to establish guidelines for managing IT services and to adjust these IT services to business requirements. IT service management thereby includes planning, monitoring and controlling the quality and quantity of IT services. ITIL is a collection of good-practice approaches that were uncovered through real-life experiences with IT service management [1].



(a) Top level ITIL processes

(b) Service design process and capacity management

Figure 5.1: ITIL perspective on capacity planning

There exist five top level processes within the ITIL framework (version 3) to develop and operate an IT infrastructure: service strategy, service design, service transition, service operation and continual service improvement (figure 5.1(a)). Capacity management as well as service level management, which is directly related to capacity management, are amongst other processes embedded into the category of service design [1]. Gunther [13] devides the capacity management process further into three sub-processes: performance monitoring, performance analysis and capacity planning as seen in figure 5.1(b). Performance monitoring covers just the collection of performance data of a system. Performance analysis is concerned about foreseeing performance problems that could possibly arise for a system and capacity planning is the process of adjusting the needed resources of a system based on the forecasts resulting from performance analysis [13].

The system developed in this thesis will base upon a component of FI-TS that is concerned with the performance monitoring process as it is fomulated by Gunther. Thus, this system represents a tool for automated performance analysis that delivers useful insights for performance planning decisions and employees involved with it. Other systems and services within an IaaS

environment, like load balancers and migration controllers for virtual machines, can request these analysis information and benefit from the gained insights.

6. Finanz Informatik Technologie Service GmbH - FI-TS

This chapter gives an introduction to the *Finanz Informatik Technologie Service GmbH* (FI-TS) by describing their cloud services (section 6.1) with a focus on their IaaS solution (section 6.2) and the currently used monitoring system for their traditional hardware landscape (section 6.3).

The FI-TS is a data center operator with headquarters in Munich, that is focussed on customers from the banking industry. It was founded in 1994, has around 500 employees and an annual business volume of around €128 million [12].

6.1. Cloud services

Since 2009 FI-TS pursues a cloud computing strategy that consists of three layers (figure 6.1): cloud.base, cloud.pla and cloud.app. cloud.base is an IaaS layer that allows customers to rent virtual machines on an hourly basis and that is comparable to Amazon's IaaS environment EC2. cloud.pla is a PaaS layer that allows customers to run applications on a standardized application environment. Hence, customers do not have to care about the servers and their setup. FI-TS has also established SaaS layer that allows customers to rent access to software services. This releases customers from operating the applications. At the current stage of development, parts of the SAP ERP suite are offered within cloud.app.

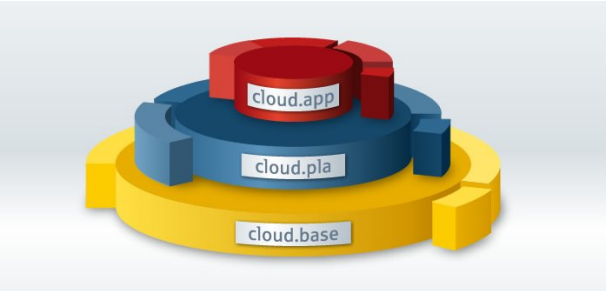


Figure 6.1: FI-TS cloud offers

6.2. Detailed look at IaaS solution

FI-TS provides Linux as well as Windows instances for its customers ranging up to 32 GiB RAM and 8 CPU cores on every instance. For now, the IaaS environment is enrolled only at the data

center in Nuremberg. This data center is certified accordingly to the ISO 27001 standard that poses requirements for "establishing, implementing, operating, monitoring, reviewing, maintaining and improving a documented Information Security Management System" [16] with respect to the organization's business risks. The IaaS environment itself was additionally certified by the TÜV Nord GmbH.

The software component that manages IaaS resources at FI-TS is called Nimbus. It has an API that is compatible to Amazon's EC2 API. This lowers the switching costs for EC2 customers to move to FI-TS.

Nimbus receives for example commands to start and stop a virtual machine (in the following: instance). Such requests are then passed to the NodeController of a physical server (in the following: node). The NodeController is deployed to every node and controls the life-cycle of the instances. The NodeController takes also care of getting access to the disk a user wants to start and mounts it to an instance. FI-TS uses the open-source hypervisor KVM that uses the Linux kernel to provide hypervisor functionalities like the scheduling of instances etc. The disk images are directly mounted from the storage backend to the instances. So they do not need to be transferred during the startup of an instance. An visualization of the system architecture of cloud.base can be found in figure 6.2.

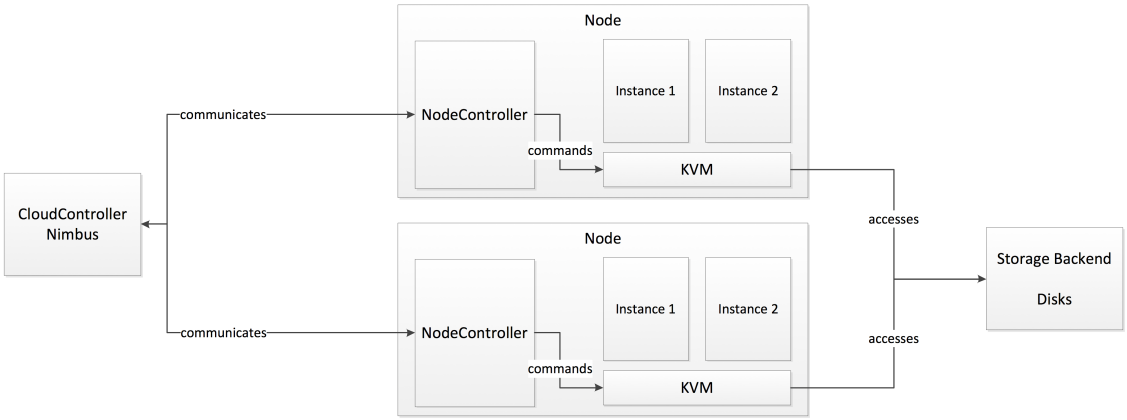


Figure 6.2: FI-TS cloud.base system architecture

As of January 2012, the first customers moved their production systems to cloud.base.

6.3. Current monitoring solution

FI-TS uses a monitoring solution called EBS (x-cellent technologies GmbH) for the servers which are not part of the IaaS environment. EBS detects performance and availability problems of servers in realtime. Therefore, threshold levels are defined that are monitored. If a threshold is reached an alarm is generated and sent to the responsible persons via e-mail or SMS. Trend evolutions can be recognized by looking at the different time resolutions the system offers for the monitoring data. EBS is not limited to monitoring hardware performance metrics, it is also able

to monitor performance data of applications, the user behaviour, IT security specific concerns (firewall entries, operations system users) and the physical infrastructure, like UPS and ambient temperature.

EBS also allows to specify dependencies between services and applications for doing a root cause analysis and alarming the right persons on the case of an outage. Another feature of EBS is the ability to create different views on business processes for the different divisions of a company. Hence, it is possible to create views customized to the needs of operations employees as well as to the needs of managers. Figure 6.3 shows two metrics that were configured for a custom view: on the left side the errors that appear in the log file of an application are visualized and on the right side the heap memory utilization of a Java application server is displayed. Users can switch between the different time spans that are available (hour, week, month, year) with tabs. Another view type supported by EBS shows the last 24 hours for a metric (figure 6.4).

From a business perspective, EBS also monitors the compliance to service level agreements that were given to FI-TS customers. This information can then be used to generate service level reports.

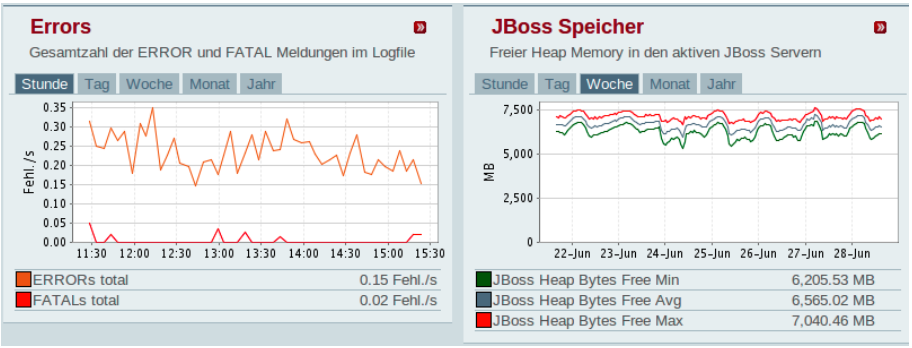


Figure 6.3: Sample dashboard in EBS

One of the shortcomings of EBS is, that it is not intended for the use in cloud environments where frequently new instances appear and where servers have to register the metrics they are measuring. Another shortcoming of EBS is that it has no automated forecasting mechanism integrated and thus, it only has limited capability to support capacity planners. Furthermore, it is not possible to hover over data points to get more details about them. These three points have been the cause for initiating a project at FI-TS for developing a first pilot application which reveals these drawbacks.

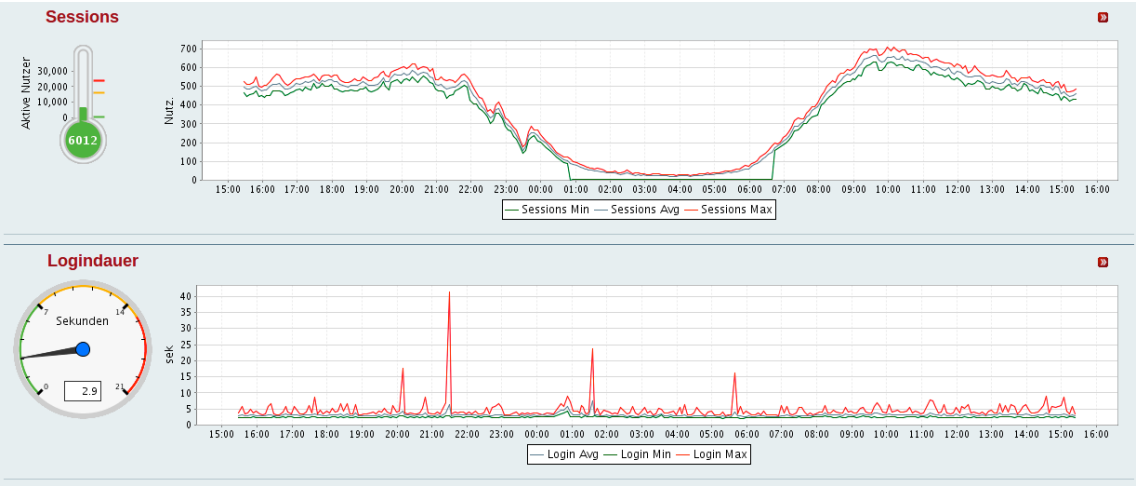


Figure 6.4: Detailed daily view in EBS

7. Cassandra

Cassandra is a column oriented noSQL database, which is implemented as a distributed hash map. It was invented at Facebook Inc. for Facebook's inbox search. Since late 2008 it is available as open-source software and many other companies like Rackspace Inc., Twitter Inc., Digg Inc. and Reddit Inc. started using and advancing it. Twitter for example uses Cassandra for real-time analytics and data mining on user data (but not for storing tweets). Rackspace uses Cassandra for persisting monitoring and logging data of its cloud services. The largest known setup of Cassandra is still located at Facebook itself and consists of around 100 servers and 150 TiB of data [15].

This chapter gives an introduction to the noSQL database Cassandra, which is used at FI-TS to store monitoring data. Firstly, the key design principles of Cassandra will be explained (section 7.1), followed by a description of the data model it uses to store data (section 7.2). Finally, Cassandra's architecture that implements the design principles will be illuminated (section 7.3).

7.1. Design principles

Shared nothing architecture
Cassandra comprises a shared nothing architecture. This means that all servers of a cluster are equal in terms of functionality and do not share state across each other. Thus, there is no notion of a master or slave within a cluster. This avoids a single-point-of-failure and minimizes the impact of outages and helps to gain high availability [15].

Seamless scalability
Furthermore Cassandra was constructed on the principle of seamless scalability which demands easy addition and removal of nodes to/from a cluster without having to manage data migrations manually [15].

Fault tolerance
Another design principle of Cassandra is fault tolerance. Even if some nodes of a cluster fail, the system should mostly be able to continue working. If a system is fault tolerant, this will positively affect the availability of the system [15].

7.2. Data model

Most of the facts provided in this section originate from the Wiki page "Data model" of the Apache Cassandra project [5].

Cassandra's data model can be described as a hash map with columns as smallest entities and containers around these columns. Such a container structure can also be found in traditional relational database systems, where the smallest entity is an attribute and several containers like rows, tables, databases are built up on them. The UML class diagram of figure 7.1 visualizes the data model of Cassandra.

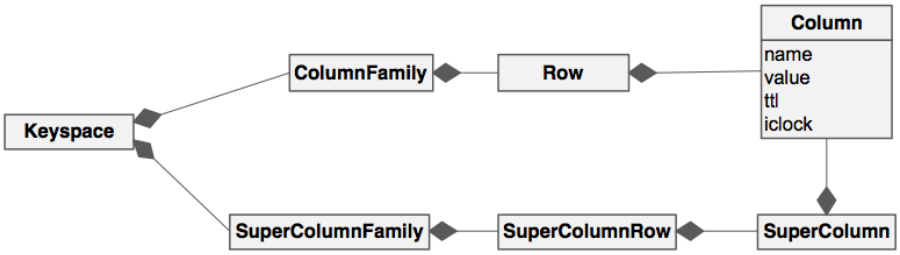


Figure 7.1: Data model of Cassandra

Column
Within Cassandra, a column is the smallest entity and a keyspace is the biggest one. A typical Cassandra column has four attributes:

- a name that identifies the column within a row,
- a value that contains the payload of a column,
- a timestamp (iclock) attribute that contains the timestamp of the last modification of the column's payload and
- a time-to-live (ttl) attribute, which states how long the column will be stored.

Row
A row is a container for columns and has a key which determines the nodes of a cluster that are responsible for storing and replicating the row. The columns that reside in a row may vary from row to row. This is a major difference to relational databases where the columns of a row are defined for a whole database table.

Column family
The next bigger containers are column families that contain an ordered collection of rows. The sort order is thereby adjustable through comparators (like it is known from Java's comparator concept). When a user creates a new column family he has to specify a comparator that should be used for the rows of the column family. Afterwards, changes to the sort order are not possible. Cassandra offers two comparators by default: "AsciiType" which assumes that row keys contain an ASCII encoded string and "UTF8Type" which assumes that row keys contain an UTF8 encoded string; but it is also possible to write custom comparators.

Keyspace
The biggest data container within Cassandra is a keyspace. A keyspace is comparable to a database of a traditional relational database. It contains several column families and has two configuration parameters: the number of replicas to store for a row ("ReplicationFactor") and

the strategy for distributing replicas across the nodes of a cluster ("ReplicaPlacmentStrategy"). A keyspace can contain 2^{127} rows at most.

Super columns
Cassandra supports yet another level of indirection. Besides standard columns, also super columns exist which contain a sorted map of columns. To enable the super column feature it has to be mentioned already at the initial setup configuration of a column family.

7.3. Architecture

In Cassandra, a feature called tunable consistency allows the database client to choose between different consistency levels. This has the effect that programmers can indirectly specify the availability requirements for the database. The reason for this is the CAP theorem.

CAP theorem
The CAP conjecture by Eric Brewer, that was raised to a theorem by Gilbert and Lynch [9], describes trade-offs between three major design goals of distributed storage systems: **c**onsistency, **a**vailability and **p**artition tolerance. Partition tolerance means that a system has to continue working even if there are message losses inbetween the nodes of a cluster. The CAP theorem states that only two out of the three properties can be chosen for a system and the third property has always to suffer. Most of the recent noSQL databases favor either availability in conjunction with partition tolerance (these systems are called AP systems) or consistency in conjunction with partition tolerance (CP systems).

Traditional relation databases guarantee the ACID properties [14]:

- Atomicity: transactions finish only in a whole or not at all
- (Strong) Consistency: data does not contain contradictions
- Isolation: concurrent transactions on the same data entity won't mess up data
- Durability: once a transaction has succeeded its changes will be available for the whole system

Hower, especially on bigger setups where relational databases are distributed over several servers it gets difficult to keep up all the ACID properties [15].

Cassandra is able to behave like a AP or a CP system, depending on the consistency level a user specifies with his read and write operations. Eventual consistency means that the data in a database might be inconsistent for a while. If an update operation is done, there might be an inconsistency window during which other readers might return the old value. Since the inconsistency window has a fixed maximum length, all other nodes in the cluster will return the updated value after this time window [6].

The choice of the consistency level has a severe impact on the availability of the database: if the user chooses strong consistency (CP system), it will be more likely that the system will suffer in

terms of availability from an outage of a few nodes. If the user chooses eventual consistency, the database will be highly available but there will also exist inconsistency windows (AP system). The most important consistency levels for write operations in Cassandra are the following [15]:

- **One**: a write operation is assumed as successful as soon as one of the nodes executed it and appended it to its commit log (eventual consistency).
- **All**: all nodes dedicated to the row key that is specified in a write operation have to append the data to their commit log and to their Memtable before the write operation will return (strong consistency).
- **Quorum**: a write operation has to succeed on the majority of nodes that are responsible for the row key within a cluster.

To gain a majority with the consistency level Quorum $\lfloor (\text{ ReplicationFactor } /2) + 1 \rfloor$ nodes must successfully write the data of a write operation. Proposing a ReplicationFactor of three, only two nodes have to complete the write operation until the client will be acknowledged about the success. If the third node is down, the other nodes will store a hint, that the unavailable node needs to execute some write requests when it comes back alive. This technique is called "hinted handoff".

Cassandra users also have to specify one of these consistency levels for read operations [15]:

- **One**: returns the data of the nearest replica
- **All**: all replicas of a column are gathered, their version is compared and the newest one will be returned to the client
- **Quorum**: waits until a quorum of nodes responded to the read request and sends the columns with the newest timestamp to the client.

Write operations in Cassandra

Cassandra is known for its high performance for write requests. This results from the fact that write requests do not require a prior read operation. The flow of actions Cassandra undertakes for write operations can be seen in figure 7.2.

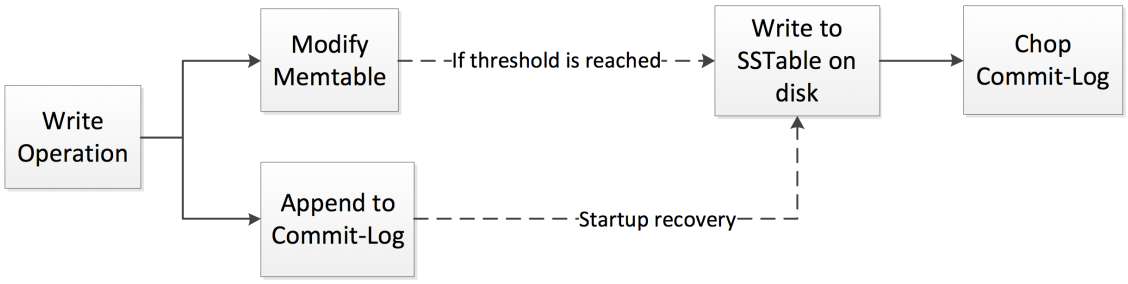


Figure 7.2: Cassandra write operation

Within Cassandra, a write operation modifies an in-memory structure called Memtable and is appended to a commit log on the hard disk. Although write operations involve a hard disk

access, they are fast because the data is only appended to the end of the commit log. If a certain threshold for the size of a Memtable is reached, it will be transformed to a sorted data structure called SSTable which is then written to the disk. After that, the data will be removed from the commit log. If the commit log is not empty at system startup, its content will be transformed to a SSTable, put to disk and the commit log will be chopped.

In order to achieve a shared nothing architecture Cassandra is based on the Peer-2-Peer concept where each peer has equal functionality.

8. Storing monitoring data

This chapter describes the requirements for a system that has to store performance monitoring data (section 8.1), as well as the infrastructure currently employed at FI-TS to comply to these requirements (section 8.2). Subsequently, the data model used by FI-TS for storing the monitoring data is overviewed and evaluated (section 8.3).

8.1. Requirements for storing monitoring data

Persistent data storage is essential for understanding the measured performance data and for the prediction of future behaviour. The size of modern IaaS data centers with several hundreds or even thousands of servers poses the following challenges to a storage system for monitoring data:

Scalability: The system has to scale well with the number of servers the data center is running. For an IaaS provider that runs 1000 servers and that measures every minute 30 different data points per server, the system already has to handle around $1000 * 30/60 = 500$ write requests per second. So new storage nodes are required to be easily added and the number of write requests the system can handle should increase linearly with the number of nodes dedicated to the storage system.

Availability: The system should be able to handle outages of a few nodes. This means that the system must not have a single point of failure and write-requests should almost always be possible. The availability of the system is especially important because several other components, like an AutoScaling component or a monitoring system for IaaS customers, in an IaaS environment depend on the performance metrics.

8.2. Current infrastructure of FI-TS

FI-TS measures performance data with a small agent on each instance as well as on each physical host (figure 8.2). Currently, the agent records the following measurements:

- CPU usage for every core
- network throughput for every network interface
- storage utilization
- storage throughput

The measurements with a resolution of one minute are published as JSON messages (a sample can be seen in figure 8.1) to ZeroMQ, a lightweight messaging system without a single point of failure. These messages contain an identifier field for the measured metric, the time of measurement, a value, additional textual information and the type of measurement. There are four types of measurements available: *gauge*, *absolute*, *counter*, *derive*. If all measurements of a metric lie in a two-sided interval the type will be *gauge*, e.g. $[0, 100]$ for cpu usage. If the measurements are unbounded, the type will be *absolute*, if they lie in a one-sided interval (e.g. $[0, \infty[$), the type will be *counter*, e.g. the overall sum of network packets sent by a network interface. If the measurements are derived locally at the agent, the type will be *derive*, e.g. network packets sent per second.

```
{"type": "GAUGE",
 "time": "2012-01-15 12:00:00",
 "value": "50.0",
 "text": "",
 "identifier": "cpu-usage"
}
```

Figure 8.1: Single measurement payload

On every Cassandra node FI-TS runs a component called collector, which receives these messages and writes them to Cassandra. The data flow is depicted in figure 8.2.

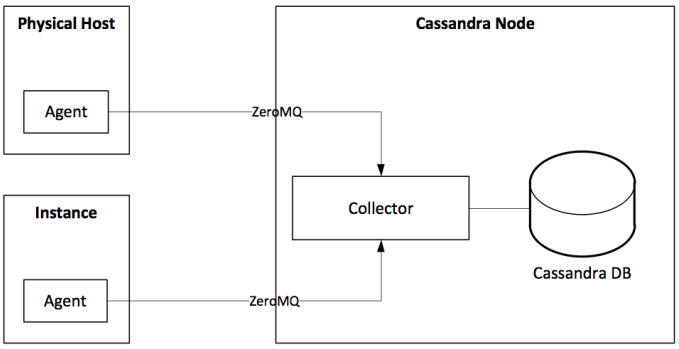


Figure 8.2: Infrastructure for transferring monitoring data to Cassandra

8.3. Data model of FI-TS

The data model of FI-TS to store monitoring data in Cassandra comprises threee column families:

- Monitoring: stores the one minute measurements.
- MonitoringAggregate: stores aggregated data points for time intervals of ten minutes, an hour, a day and a week.
- MonitoringIdentifier: keeps track of the metrics that are measured for each host.

In the following the notation of E. Hewitt's book "Cassandra: The Definitive Guide" is used to describe the data model of FI-TS. This notation is based on standard UML class diagrams. Class names indicate the name of a column family and attributes represent the name of a column. Additionally, some special UML stereotypes are used for representing some of Cassandra's concepts:

- <<CF>>: column family
- <<SCF>>: super column family
- <<RowKey>>: row key

Figure 8.3 visualizes the data model of FI-TS for Cassandra. The attributes with a hash character (#) as prefix denote that this attribute is an index field and the suffix describes the structure of its column name.

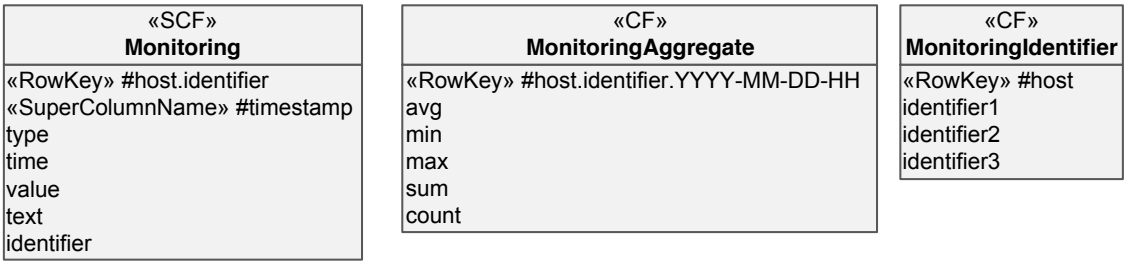


Figure 8.3: Data model to store monitoring data

The measurements that arrive every minute at the collector are stored in the Monitoring SCF under a row key. This row key consists of the hostname from which the measurement originates and the identifier of the measured metric (e.g. "hostname.identifier"). Under this row key several super columns for every measurement exist. Thereby, the super column name contains the timestamp of a measurement.

Assuming that the message of figure 8.1 originates from the server *samplehost* and that it contains the measurement of its cpu utilization at the timestamp "2012-01-15 12:00:00", the storage structure of this message looks like it is shown in figure 8.4 as JSON object.

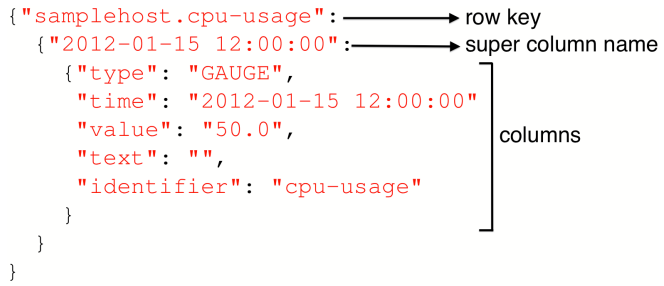


Figure 8.4: Monitoring SCF sample row

In addition to writing the one minute measurements to the database, also aggregates for longer time periods are kept up-to-date. These aggregates that include the minimum, maximum, the number of aggregated observations and the sum of measurements within the aggregation period are stored in the column family MonitoringAggregate. Currently, four aggregation levels are defined: ten minutes, an hour, a day and a week. The row key of the column family MonitoringAggregate has a similar structure as the row key in the Monitoring SCF. The row key is just extended by a suffix with the format "yyyy-MM-dd-HH-mm". Thus, the different aggregation levels can easily be identified by the start time of their period.

At the arrival of the sample measurement shown in figure 8.1 the columns under the following row keys of the MonitoringAggregate CF are updated:

- **samplehost.cpu-usage.2012-03-15-12-00**, which contains the ten-minute aggregate for the measurements between 15th March 12:00 and 12:10 (exclusive)
- **samplehost.cpu-usage.2012-03-15-12**, which contains the hourly aggregated measurements between 12 and 13 o'clock (exclusive)
- **samplehost.cpu-usage.2012-03-15**, which contains the daily aggregated measurements between 15th and 16th of March (exclusive)
- **samplehost.cpu-usage.2012-KW12**, which contains the weekly aggregated measurements of the calendar week 12

The MonitoringIdentifier CF is a simple lookup table for keeping track of the metrics that are available for a host. Assuming that there are two metrics (CPU usage and io throughput) measured for the server *samplehost*, the row within the MonitoringIdentifier CF looks as follows:

```
{"samplehost":
  ["cpu-usage", "io-tput"]
}
```

Figure 8.5: MonitoringIdentifier CF sample row

8.4. Evaluation

Designing a storage schema for noSQL databases like Cassandra is rather different from the design of a schema for traditional relational databases, because it begins with identifying queries. The schema is then deduced and optimized from these queries [15]. In order to evaluate the current data schema of FI-TS, this course of action is pursued. By looking at the requirements described in chapter 10, these queries were identified:

1. get a list of performance metrics that are collected for a server
2. get the data of a certain performance metric and a certain host within a time range
3. get a list of all hosts within the system

The fist query can be realized with a simple get command with the hostname as parameter on the data schema shown in figure 8.3. From the perspective of Cassandra this query can easily be executed because it just involves hashing the hostname, determining the nodes for this row

Time Resolution	Retention Time
1 minute	14 days
10 minutes	30 days
hour	90 days
day	365 days
week	7 years

Table 8.1.: Retention time for the different time resolutions of data

key and reading from these nodes.

The execution of the second query differs for one minute data and aggregated data (10 minute data, hourly data etc.). For the one minute data the Monitoring SCF is queried with Cassandra's SliceRange feature which allows to fetch a range of columns by their column name. In this case the SliceRange has to contain the start and end time of the data in question. Cassandra is able to execute this query in a fast manner, because the super columns within a row are sorted by their name. The query for aggregated data has to use the MonitoringAggregate CF. In this case the client application has to generate the row keys that lie between the start and the end time (e.g. samplehost.cpu-usage.2012-03-15 - samplehost.cpu-usage.2012-05-15) and fetch these rows with a multiget command. Even if the request contains several hundreds of data points this query can be executed efficiently by Cassandra.

The third query cannot be accomplished efficiently on the data model of FI-TS. It would require a full scan over the MonitoringIdentifier CF to get all hosts within the system. Furthermore, the MonitoringIdentifier CF stores only hostnames of servers, but no meta information about them. Hence, there is no change to distinguish a physical node from a virtual instance. Therefore, it was decided to query the cloud controller API of FI-TS (Nimbus) to get information like this.

FI-TS uses the ttl attribute of columns to specify a certain retention time for data before it gets deleted (see table 8.1). This is very comfortable because it reduces the growth rate for the amount of stored data. With this feature it is not necessary to periodically scan over large parts of the database to find and delete outdated data.

All in all, the FI-TS data schema is already mature because the characteristics of Cassandra have already been considered in early stages of the development of the component that collects and persists performance measurements. The problems with the third query were fixed easily by accessing an already existing API within FI-TS. Another drawback of the data model of FI-TS is that there is currently no association between a metric and its allowed range of values. For this reason, the system does not know that measurements and forecasts for the CPU usage metric may only range from 0 to 100 percentage. And thus, the forecast algorithm is not able to consider this and it might happen that it delivers data points with a negative value for the CPU usage. A solution for this problem is to introduce a new column family in Cassandra that maps the different metrics to their allowed range of values but this is not covered from the implementation part of this thesis.

9. Statistical methods for capacity planning

This chapter introduces the statistical methods used in the developed software component. At first, the method of time series analysis is explained (section 9.1), which is the basis for autoregressive models (section 9.2) and forecasting. Subsequently, two methods of descriptive statistics are explained: histograms and cumulative frequency charts (section 9.3).

9.1. Time series analysis

At first, the ambiguity of the term "time series" has to be addressed. On the one hand, a time series is a set of observations x_t ($t = 1, ..., T$), with time records for each measurement. x_1 indicates the first and oldest measurement and x_T the newest one. On the other hand, a time series also denotes the process inbehind of the data, that generated the observations. In this work both meanings will appear and it should be clear from the context which one is meant.

Conventional statistical inference deals with the derivation of conclusions from a data population based on samples [23]. A central assumption in parametrical statistics are independent and identically distributed observations. This criterion is often violated in time series analysis since the observations are time-correlated and thus dependent [19].

Formally spoken, a time series is a set of observations $\{x_t\}$ which is seen as realization of random variables $\{X_t\}$. A **time series model** is the specification of a joint distribution of these random variables X_t. **Time series analysis** tries to find an appropriate time series model for a time series of data [3]. In the following (also in the implementation), the considerations are restricted to time series data with equally sized distances between the observations.

Robert Shumway and David Stoffer [19] describe three types of time series models apart from classical regression: autoregressive integrated moving averages models (called ARIMA-models), additive models and frequency models. Classical regression in time series analysis explains the observations x_t as linear combination of a fixed set of inputs. Thereby, a constant error variance of the samples over time is assumed [19]. **ARIMA** models try to explain observations with a fixed number of preceding observations of x_t. Several branches for ARIMA-models are commonly used: autoregressive models (AR) that use the last p measurements in a regression model, moving average models that use the moving average of the last p measurements. The combination of these two models is called ARMA model. ARMA models can optionally be extended with an integrational part of the last measurements; then they are called ARIMA-models [19]. With additive models the observed data is tried to be explained by a sum of different influences. Frequency models assume that the main characteristics of a time series can be explained by periodic variations. Thereby, the time series is often decomposed into different frequencies [19].

9.2. Autoregressive models

An autoregressive model of order p, often referred to as AR(p), explains a point of a time series x_t by means of the p last observations $x_{t-1}, \dots x_{t-p}$ that are direct predecessors of x_t (equation 9.1) [19].

$$x_t = \sum_{i=1}^{p} \phi_i x_{t-i} + \epsilon_t \tag{9.1}$$

The parameters ϕ_i of the model have to be estimated. This can be done using ordinary least squares or using the Yule-Walker-Equations. The error term ϵ_t is assumed to be Gaussian white noise. This means that ϵ_t has to follow a gaussian distribution with mean zero and a finite variance. Another requirement for applying this model is, that the observations x_t have mean zero; if this is not the case, the mean (μ) has to be subtracted from every occurence of x_t, resulting in the following equation [19]:

$$x_t - \mu = \sum_{i=1}^{p} \phi_i (x_{t-i} - \mu) + \epsilon_t \tag{9.2}$$

Expanding equation 9.1 will result in the following set of linear equations:

$$\begin{pmatrix} x_{t-1} & x_{t-2} & \dots & x_{t-p} \\ x_{t-2} & x_{t-3} & \dots & x_{t-1-p} \\ \vdots & \vdots & & \vdots \\ x_{t-n} & x_{t-n-1} & \dots & x_{t-n-p} \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_p \end{pmatrix} = \begin{pmatrix} x_t \\ x_{t-1} \\ \vdots \\ x_{t-n+1} \end{pmatrix} \tag{9.3}$$

To solve these equations and to get the model parameters ϕ the Yule-Walker method is used. Therefore each row is muliplied by x_{t-d}, where d indicates the row index. So the first line will be multiplied by x_{t-1}, the second by x_{t-2} etc. The Yule-Walker equations (equation 9.3) result from normalising with the expected values for each row. The equations now look as it is shown in equation 9.4. Thereby, $\rho(\tau)$ denotes the autocorrelation coefficient at delay τ [2].

$$\begin{pmatrix} 1 & \rho(1) & \rho(2) & \rho(3) & \dots & \rho(p-1) \\ \rho(1) & 1 & \rho(1) & \rho(2) & \dots & \rho(p-2) \\ \rho(2) & \rho(1) & 1 & \rho(1) & \dots & \rho(p-3) \\ \vdots & \vdots & & & & \vdots \\ \rho(p-1) & \rho(p-2) & \rho(p-3) & \rho(p-4) & \dots & 1 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_p \end{pmatrix} = \begin{pmatrix} \rho(1) \\ \rho(2) \\ \vdots \\ \rho(p) \end{pmatrix} \tag{9.4}$$

To understand the autocorrelation coefficient, the autocovariance function has to be introduced. The autocovariance describes the covariance of a time series x_s and a shifted version of it x_t [3]:

$$\gamma(s,t) = \text{cov}(x_s, x_t) \tag{9.5}$$

The delay between the two time series is denoted with $\tau = s - t$. After normalizing the autocovariance function with the variances of the two time series the autocorrelation function (ACF) can bedefined and the autocorrelation coefficient at delay τ results [3]:

$$\rho(s,t) = \frac{\gamma(s,t)}{\sqrt{\gamma(s,s)\gamma(t,t)}} = \rho(\tau) \tag{9.6}$$

With the transformation of the linear equation 9.3 to equation 9.4, the linear equation system is simpler to solve. By solving this equation system, the parameters of the autoregressive model ϕ result.

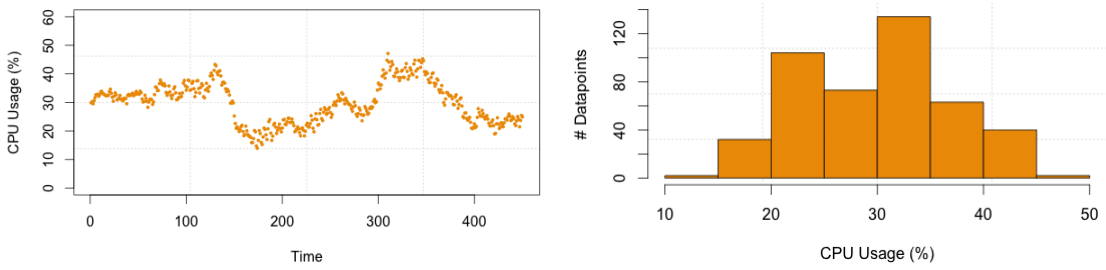
After the estimation of model parameters ϕ, it is possible to calculate forecasts for a time series. To calculate the forecast value at time $t + 1$, the p preceding values of x_{t+1} have to be multiplied with the respective parameter of the model in accordance to equation 9.1:

$$x_{t+1} = \phi_1 x_t + \phi_2 x_{t-1} + \cdots + \phi_p x_{p-1} \tag{9.7}$$

9.3. Histograms and cumulative frequency charts

Histograms are widely used in descriptive statistics to get a rough sense for the density and hence for the distribution of observed data. Therefore, the co-domain of the data is devided into several equally sized bins and the number of data points represent the might of a bin. As example a plot of 450 sample data points as well as its corresponding histogram with eight bins is shown in figure 9.1. The histogram reveals that the majority of datapoints of this example lie between 20 and 35, with a peak of over 120 (25 percent of all data points) which belong to the bin that ranges from 30 to 35.

For capacity planners histograms are of use because they show in figure 9.1 that 80% of the measurements for the CPU usage of a server lie in the interval $[50, 60]$. This means that the server has a mid-ranged CPU consumption most of the time and it could be possible to hand over some of its CPU resources to other instances on the same node.



(a) Sample Data

(b) Histogram of sample data

Figure 9.1: Histogram example

Cumulative frequency charts are based upon histograms. They cumulate the bars of a histogram from the left to the right and describe the cumulative density function that underlies the data.

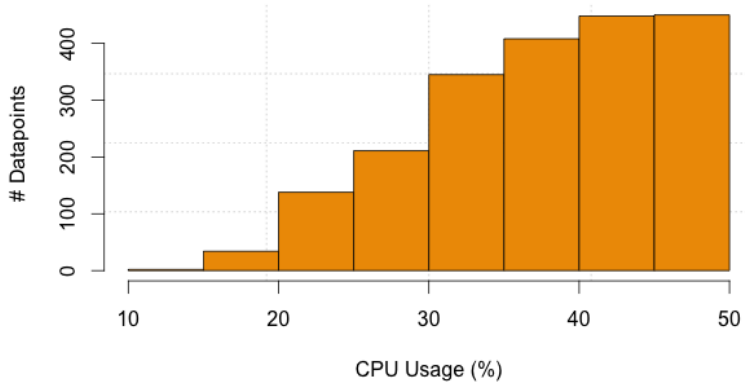


Figure 9.2: Cumulative frequency chart example

The bar at the right border of figure 9.2 shows that all 450 data points lie below the value of 50. The second bar from the right denotes that almost all data points lie below the value of 45.

From a capacity planner's perspective, the cumulative frequency chart shown in figure 9.2 reveals that an instance suffices 50% of its allocated CPU resources for 90% of the time.

Part III.

Concept

10. Requirements engineering

Requirements engineering is concerned about defining the requirements of a software component and consists of two activities: requirements elicitation and requirements analysis. Requirements elicitation covers the identification of the functional and nonfunctional requirements of a software project. It is also helpful for establishing a consistent terminology between the client and the contractor of a software project. During requirements analysis these requirements are put into an analysis model [4].

This chapter will cover both activities for the software component that was developed during this thesis. The requirements elicitation is described in section 10.1 and the requirements analysis in section 10.2.

10.1. Requirements elicitation

This section describes the functional (section 10.1.1) and nonfunctional requirements (section 10.1.2) that were elicited with the stakeholders Christian Brunner and Stefan Majer from FI-TS and Matheus Hauder from the sebis chair of the Technical University of Munich.

10.1.1. Functional Requirements

Functional requirements describe the functionality which different stakeholders within a software engineering project expect from the system once it has been finished. For the capacity planning tool that has been developed within this thesis, these functional requirements were elicited by the stakeholders:

FR01 The system has to show an overview of the workload of the complete IaaS environment of an IaaS provider.

FR02 The monitoring data of performance metrics have to be visualized for a single server as well as for a group of servers. Thereby, users have to be able to choose a time resolution and a time interval, they want to explore.

FR03 The different performance metrics that are available for a single server and for a group of servers have to be listed.

FR04 The system has to find threshold violations in the monitoring data.

FR05 The system has to make forecasts for the future development of performance metrics of a single server and also for a group of servers.

FR06 Additional statistics about the monitoring data, like histograms and cumulative frequency charts have to be displayed.

FR07 Users have to be able to search for servers and for groups of servers.

FR08 An IaaS-Customer has to be able to request monitoring data for his instances.

FR09 External systems like auto scaling systems or load balancers have to be able to request monitoring data as well as forecasts for them, so that they can consider these informations in their decisions.

10.1.2. Nonfunctional requirements

Nonfunctional requirements constraint the development of a software component without posing new functionality to the system. The following nonfunctional requirements were elicited for the constructed system:

NR01 The component, that queries domain level objects from an IaaS provider, should be adaptable for other IaaS providers.

NR02 The calculation of forecasts has to scale well with the number of servers of an IaaS provider.

NR03 The statistical model to calculate the predictions has to be exchangeable.

NR04 A single forecast has to take less than a second.

NR05 The storage system that persists the performance data has to be exchangeable.

10.2. Requirements analysis

10.2.1. Use case model

The first step in modeling the functional requirements with an UML use case diagram is to identify the different actors that are involved in the system. Since the system is developed for capacity planners and operations employees, it is clear that these two user groups have to be considered as actors.

Another type of actors within the system are the customers of an IaaS provider. They have only restricted rights for the system and should only be able to query the monitoring data of their servers but not the threshold violations. The last group of actors, are external systems that run within the network of an IaaS provider. They query monitoring data to enhance their decision making and need only access to the monitoring data as well as to the threshold violation data. The visualizations are not of interest for these systems.

After the identification of the actors the functional requirements of section 10.1 are mapped to the following use case names:

- **FR01** is mapped to use case “visualize monitoring data”
- **FR02** is also mapped to use case “visualize monitoring data”

- **FR03** is mapped to use case “list metrics”
- **FR04** is mapped to use case “get threshold violations”
- **FR05** is mapped to use cases “get forecasts” and “ visualize forecasts”
- **FR06** is mapped to use case “get histogram and cdf”
- **FR07** is mapped to use case “search”
- **FR08** is mapped to use case “get monitoring data”
- **FR09** is mapped to use cases “get monitoring data” and “get forecasts”

The three types of actors that were identified and their access rights for the use cases can be drawn as an UML use case diagram, which is shown in figure 10.1. The UML stereotype “extends” which is used for the association between the actor “External System” and “Customer” denotes that all use cases that can be initiated by an IaaS customer may also be initiated by an external system.

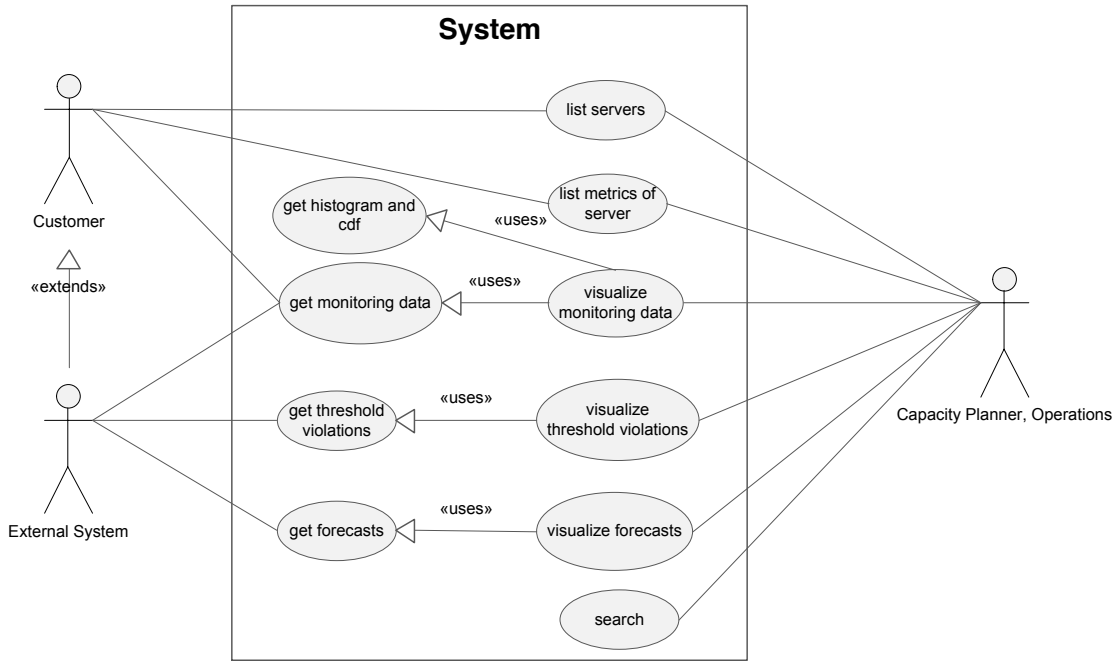


Figure 10.1: Use case diagram

10.2.2. Static model

To avoid misunderstandings between the different stakeholders of a project it is useful to clarify the terminology the software component will use and what the responsibilities of important analysis objects are. Therefore, the participating objects are identified for each use case [4]. These objects are initial object-candidates for a static object model. The use cases of figure 10.1 have these participating objects:

- **visualize monitoring data:** MonitoringData, Server, TimeResolution, TimeSpan
- **get monitoring data:** MonitoringData, Server, TimeResolution, TimeSpan
- **get histogram and cdf:** Histogram, CDF
- **get forecasts:** Forecast
- **visualize forecasts:** Forecast
- **list metrics of server:** Metric, Server
- **list servers:** Server
- **get threshold violations:** ThresholdViolation
- **visualize threshold violations:** ThresholdViolation
- **search:** -

After that, a refinement step is done in order to get to a static object model. The initial object MonitoringData is split up into two objects: TimeSeries and DataPoint. The Forecast object is renamed to Prediction and a generalization is introducted for the objects Node and Instance.

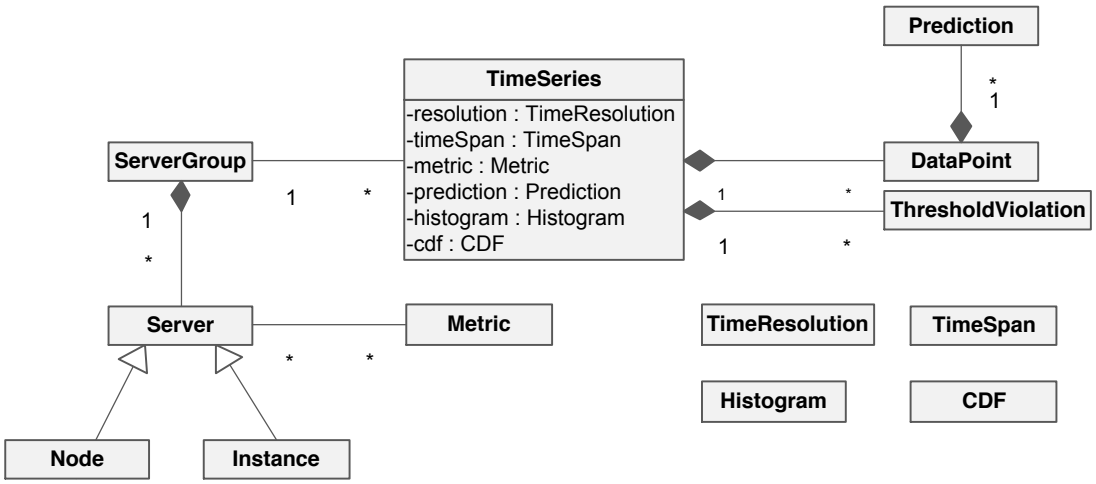


Figure 10.2: Initial static object model

The responsibilities of each of the objects of the static object model (figure 10.2) are explained in the following:

TimeSeries: consists of DataPoints, ThresholdViolations and optionally a Prediction; they belong to a Server or a group of servers.

DataPoint: represents one measurement of a metric for a certain time.

Metric: represents the type of measurement, e.g. CPU usage or I/O throughput. Each Server has several metrics that are measured and each TimeSeries belongs to a certain Metric.

TimeResolution: represents the time resolution of a TimeSeries object. Actors have to specify a TimeResolution on the request for a TimeSeries. Only a static set of time resolutions is available: one minute, ten minutes, hour, day and week. If an actor specifies a resolution

of 10 minutes, he will get a TimeSeries object where the data points of the time series have a distance of 10 minutes.

TimeSpan: represents the time span (start and end date) of a TimeSeries.

Histogram: contains the histogram for the datapoints of a TimeSeries.

CDF: contains the cumulative frequency chart of the datapoints of a TimeSeries object.

Prediction: is a wrapper object that contains the DataPoints that have been predicted for a TimeSeries object as well as accuracy measures.

ThresholdViolation: represents an event where a threshold was reached. A TimeSeries may contain several ThresholViolation objects depending on how often a threshold has been reached within the corresponding TimeSpan.

Server: represents either a node or an instance. Nodes host a couple of instances.

ServerGroup: represents a group of servers.

For reasons of clarity, the figure 10.2 misses the one-to-one relationships between the TimeSeries object and the objects TimeResolution, TimeSpan, Histogram and CDF.

Every IaaS environment has a taxonomy which is built up on the smallest infrastructure entity - a virtual machine instance. In figure 10.3 the taxonomy that is used by FI-TS is shown. This taxonomy is also valid for most parts for Amazon EC2. In this model a customer represents an institution that buys services from an IaaS provider. A customer and an IaaS provider conclude a contract over the services that should be available for the customer. A contract may contain several security groups. A security group contains several instances and has its own firewall rules. By default, it is not possible to access instances of a security group from other security groups. Scaling groups establish another level of indirection between security groups and instances. A scaling group is a group of instances with an elastic size which may automatically adapt the number of instances it consists of, accordingly to a load measure like the number of parallel sessions which a web application faces.

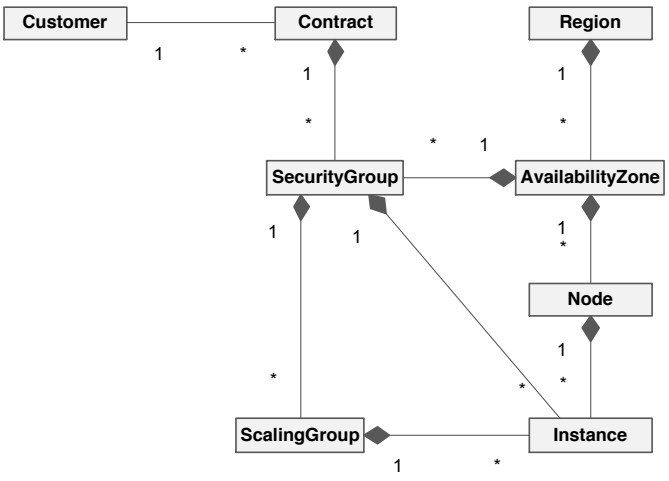


Figure 10.3: Common IaaS taxonomy as class diagram

From the physical point of view, an instance runs on a node that is located in a specific availability zone. An availability zone is an isolated data center section, i.e. mostly a separated room with a fireproof door. Nodes can be grouped by an availability zone, region and at a global level. Instances can be grouped by their scaling group, their security group, their contract and their customer.

To illustrate this taxonomy with an example, the online banking application of the "Directbank" will be used in an UML object model (figure 10.4). The "Directbank" runs their online banking application at FI-TS. A special security group exists for this contract to isolate the application from other customers. The security group consists of native instances that host the database as well as of a scaling group (WebServerScalingGroup) for the web and application servers. For example purposes, it is assumed that there exist three web servers within the scaling group. According to the IaaS paradigm, it is possible that a web server instance runs side by side on the same node as a database server instance (this is the case for node3). All nodes that are concerned with online banking are gathered in an extra availability zone (OLBZone). The availability zone is located at the data center in Nuremberg.

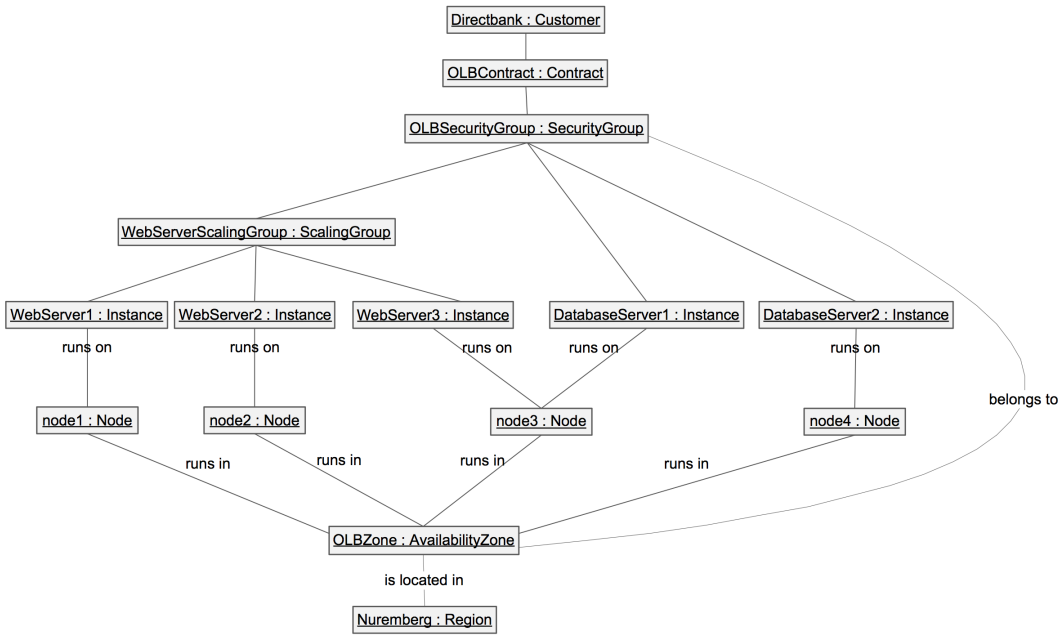


Figure 10.4: Sample instance of the FI-TS IaaS taxonomy as object diagram

11. System and object design

System design is the first modeling step in software engineering projects for introducing internal structures of a system into its models. This step covers the decomposition of the system into smaller and better manageable parts (section 11.1), called subsystems as well as the distribution of the software components to different servers [4]. In section 11.2 the different strategies that were considered for implementing the forecast functionality are described. Afterwards, services within these subsystems are identified and interfaces are designed for them (section 11.4). In section 11.3 mockups for the user interface are shown.

11.1. Subsystem decomposition

The aim of decomposing a system into several smaller subsystems is to reduce its complexity in the manner of the divide and conquer principle. During the design of a subsystem decomposition two properties are of major interest: the degree of coupling, which describes the number of connections between subsystems and the degree of cohesion, which describes the number of connections between the inner parts of a subsystem. An ideal subsystem decomposition can be characterized as having a low degree of coupling and a high degree of cohesion. If these goals are maximized, a set of relatively independent subsystems results, where modifications in one subsystem have just little impact on other parts of the system [4].

For the capacity planning support tool that was developed during this thesis, six subsystems were identified: GUIClient, GUIServer, TimeSeries, ThresholdViolation, Storage and IaaSAdapter (see figure 11.1). The architecture of the subsystem hierarchy follows a closed layer architecture, where each layer may only access the layer immediately below it. Thereby, three layers were identified: a GUI layer that capacity planners and operators will see (GUIClient, GUIServer), a logic layer that contains application logic (includes the components TimeSeries and ThresholdViolation) and a data layer that is concerned with the data access (Storage and IaaSAdapter).

The **GUIClient** is responsible for laying out the UI elements like buttons and textareas etc., as well as for sending requests to the GUIServer to receive data from it and to fire events for actions, a user performs on the UI. The **GUIServer** subsystem is responsible for handling these requests. It uses services provided by the subsystems TimeSeries and ThresholdViolation to gather necessary information. Functionality around time series data and operations on them are gathered in the subsystem called **TimeSeries**. It accesses information provided by the IaaSAdapter and Storage subsystem. The **ThresholdViolation** subsystem is concerned with detecting threshold violations within time series data. To interfere with the domain model of an IaaS provider a dedicated subsystem called **IaaSAdapter** was introduced. Functionality for accessing monitoring data is bundled within the **Storage** subsystem.

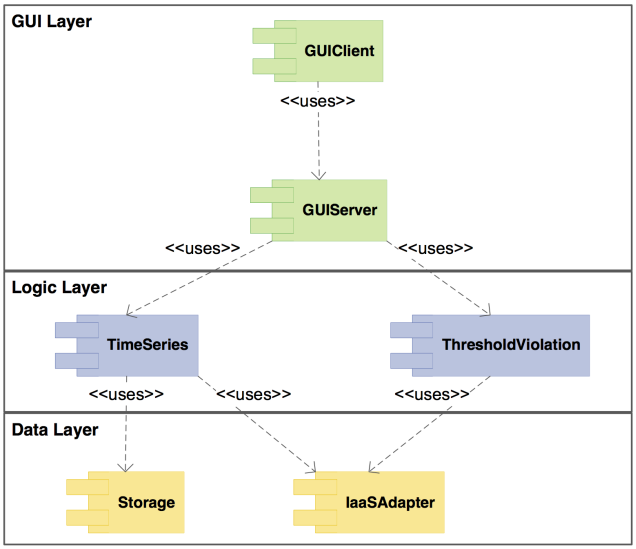


Figure 11.1: Component diagram

Hardware software mapping

After decomposing the system into subsystems, the next question that arises is how to map the different parts of the system to hardware components. It is obvious that the GUIClient will run within the browser of a website user. The rest of the system (GUIServer, TimeSeries, ThresholdViolation, IaaSAdapter and Storage) will be mapped to a Java servlet container. In figure 11.2 this mapping is shown with an UML deployment diagram. This diagram contains also the nodes and components which the system will communicate to in the case of a deployment at the FI-TS. In this case the IaaSAdapter subsystem has to communicate with the controller subsystem of the IaaS environment ("NimbusController") and the Storage subsystem has to communicate with the Cassandra database. For this subsystem the MonitoringDAO of FI-TS is reused.

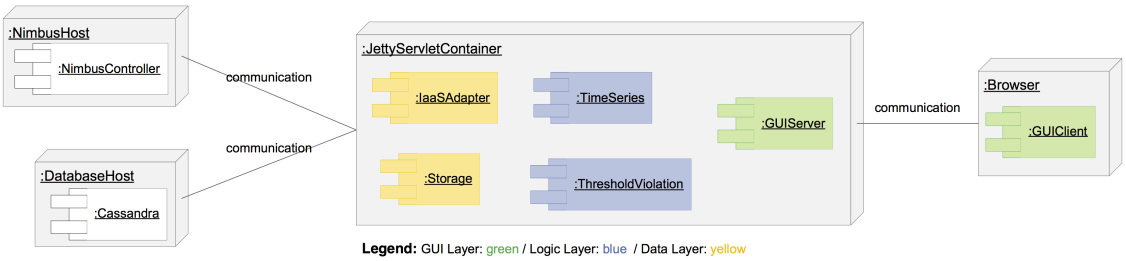


Figure 11.2: Deployment diagram

11.2. Strategies for implementing the forecast functionality

For the implementation of the forecast mechanism, four different strategies have been considered and evaluated.

Simple batch process
The simplest strategy is a batch process that iterates over all servers and calculates forecast values for each of the servers' metrics. This is computationally very expensive because forecasts for metrics that are irrelevant at this point at time are also calculated.

Batch processing with configuration
The batch-process strategy can be extended by a mechanism to configure metrics that are considered during the batch-process run. But with this strategy a system to configure these metrics is needed which yields in higher implementation efforts.

Streaming approach
The third strategy is based on streaming data and adjusts the forecasts for a metric when new measurements arrives. But this strategy has the same drawbacks as the simple batch-process: there are many irrelevant metrics and it does not make sense to calculate forecasts for every metric.

Request based approach The last considered strategy is request-based. This means that the forecasts are computed during an HTTP request that arrives at the server. The implementation efforts for this strategy are low and if the forecasts can be calculated within reasonable time (< 0.3 seconds), this will be acceptable for users of the system. Since a first implementation of autoregressive models showed that forecasts can be calculated within reasonable time, this strategy was chosen for the implementation.

11.3. Mockups

Mockups describe the layout of UI elements with an abstract drawing. They show the navigation paths and the accessibility of information of the different pages of a website. But the general concept of mockups is not limited to webdesign only. It is also used in other engineering diciplines to get a rough sense of how a system or device will look and feel like at an early stage of a project.

For the system under construction, several mockups were developed. A mockup for viewing information directly related to an instance, a node, an availability zone, a region, a scaling group, a security group, a customer, a customers contract and a global view over all datacenters exists. In the following only three of them are described in detail: the global (subsection 11.3.2), the region (subsection 11.3.3) and the instance view mockup (subsection 11.3.4). These are the most important ones for the user group of the system. Additionally, the layout that all mockups have in common is described in subsection 11.3.1. The other mockups can be found in the appendix chapter C. A sitemap for the different views the system comprises is depicted in figure

11.3. This reflects also the most important navigation and click paths of a user. Starting-point of the system is the global view where the customers and regions of a datacenter provider are listed. From there the user may navigate to the view of a certain customer or region.

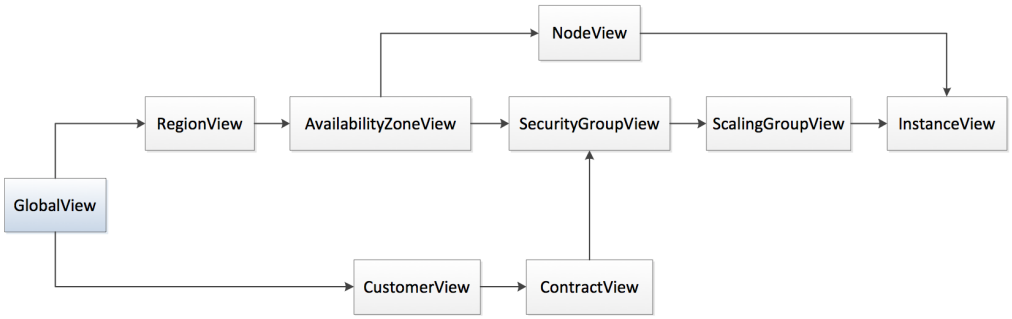


Figure 11.3: Sitemap and click paths

11.3.1. Abstract mockup

All the mockups have a fixed layout of content and navigation sections in common. This abstract layout is depicted in figure 11.4. Sections that are concerned with navigation purposes have a blue background color in this figure. The navigation section at the top uses the UI design pattern of breadcrumbs. A breadcrumbs navigation shows "[...] each level of hierarchy leading to the current page, from the top of the application all the way down. In a sense, they show a single linear slice of the overall map of the site" [22]. They are very practical for showing users where they are relative to the whole site [22].

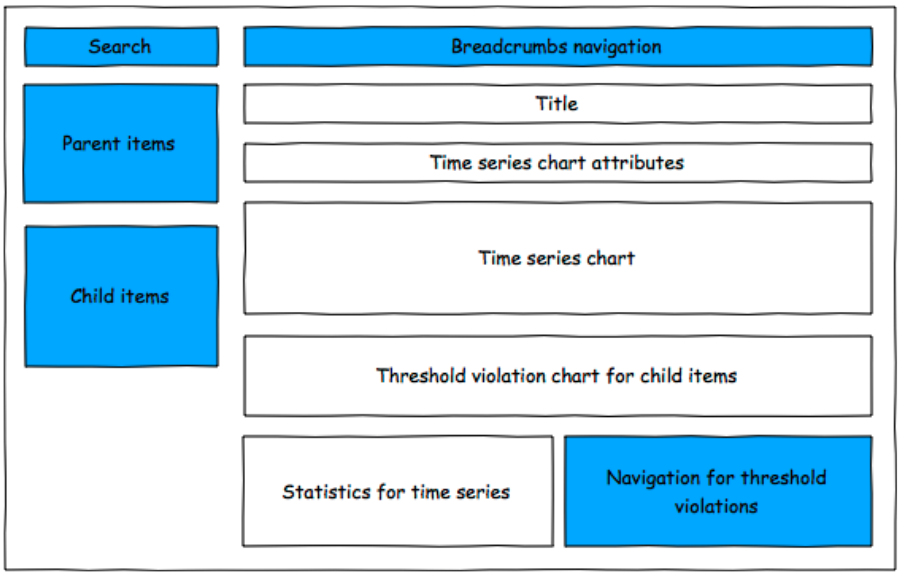


Figure 11.4: Abstract mockup

Threshold level	Color
low	green
medium	yellow
high	red
critical	black

Table 11.1.: Mapping between threshold violation levels and colors

If a user opens the instance view, the breadcrumbs navigation shows a link to the scaling group, security group, availability zone and region the instance belongs to (e.g. Global / Region: Nuremberg / Availability Zone: Nuremberg AZ-1 / Security Group: OLB / Scaling Group: OLB Web Scaling Group / Instance: nginx1).

The second navigation section is called "Parent items" and lists elements that are related to the currently displayed item. Underneath the "Parent items" section the child elements of the current item are listed ("Child items" section). Users can easily jump to one of the pages without having to navigate the whole path down to them by using the search field which is provided in the search section. The title section of figure 11.4 shows the current page title. For letting users specify the metric, time resolution and timespan of a time series, the "Time series chart attributes" section was introduced. The "Time series chart" section underneath contains the time series chart. Thereby, the abscissa is a time axis and the ordinate shows the resource utilization for a metric. Thresholds for the currently displayed item (e.g. an availability zone) are drawn directly into the time series chart as constant functions. This simplifies the identification of threshold violations within a time series. Below the time series chart, the threshold violations of the child items for the currently displayed item are visualized ("Threshold violation chart" section). At the bottom of the page additional statistical information like the histogram, cumulative frequency chart as well as the mean and standard deviation of the datapoints of the time series are displayed ("Statistics for time series" section). If a user clicks on one of the datapoints of the threshold violation chart, the section to the right shows detailed information about the threshold violation ("Navigation for threshold violations" section).

Threshold violation chart
The question that arises is how users are able to drill down from an upper level threshold violation (e.g. an availability zone threshold) to the cause of a problem which is mostly found in the lower level entities (like instances). To solve this navigation problem an interactive scatter plot is used (figure 11.5). It shows the threshold violations of the child elements, like the security groups and nodes that belong to an availability zone. Thereby, the abscissa denotes a time axis (which is shared with the upper time series chart) and the ordinate denotes an axis for the different thresholds. Because the scatter plot is placed directly under the time series chart and because they share a common abscissa, it is easy for the human eye to find correlations between the two plots [22]. Additionally, colors are assigned to the different threshold levels to simplify the distinction between them (table 11.1). The size of the data points in the scatter plot increases with the number of violations of the same level that take place at the same time.

If a user notices a threshold violation for an availability zone in the time series chart, he just has to look down to the scatter plot and search for threshold violations of high priority nearby. With a click on one of these points the user will get details about the datapoint in the "Navigation

for threshold violations" section. For each entity that is involved with this datapoint, a link is displayed in the navigation section for threshold violations so that a user can further drill down to the cause of a problem.

An alternative to the scatter plot visualization is drawing a bar chart for every child item of a page which shows the temporal development of its threshold violations. This approach was considered at early stages of the thesis but it is not practical anymore if there exist several hundreds of child items; e.g.: if there exist several hundreds of customers, the GlobalView would have to display for all of them an extra chart for the thresold violations. This will not fit onto a single page and thus, it is difficult for users to overlook and check all of theses charts. This is the reason why this approach has been discarded.

Figure 11.5: Visualization of threshold violations

11.3.2. Global view mockup

The global view (figure 11.6) is the starting-point for users of the system. It shows an overview of the aggregated resource utilization over all regions of an IaaS provider and thus, is important for capacity planners to see the development of the global resource utilization. The threshold violation chart displays the threshold violations of all regions and all customers. The "Child items" section holds two subsections for listing the regions and the regions and the customers an IaaS provider has.

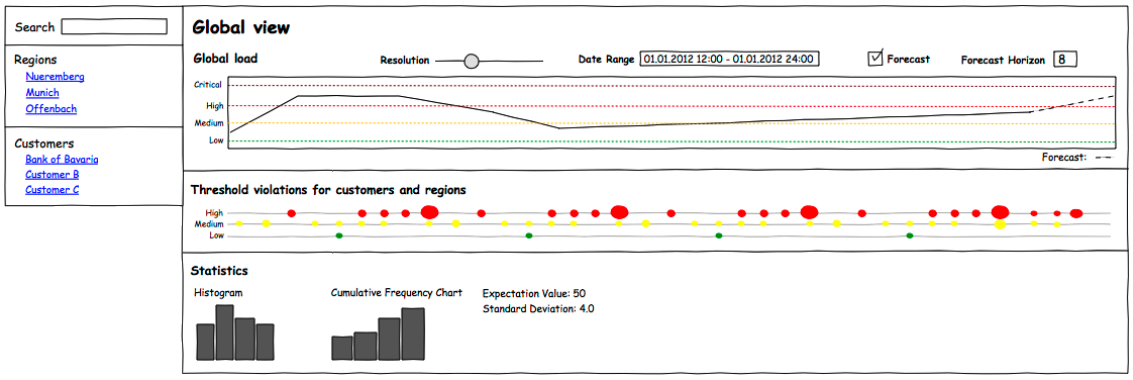


Figure 11.6: Global view mockup

11.3.3. Region view mockup

The region view shows the aggregated resource utilization of a whole region of an IaaS provider. Thereby, the threshold violation chart contains the threshold violations of the availability zones

of a region. A list of all availability zones of a region is also shown in the "Child items" section. The mockup that was created for this view can be seen in figure 11.7.

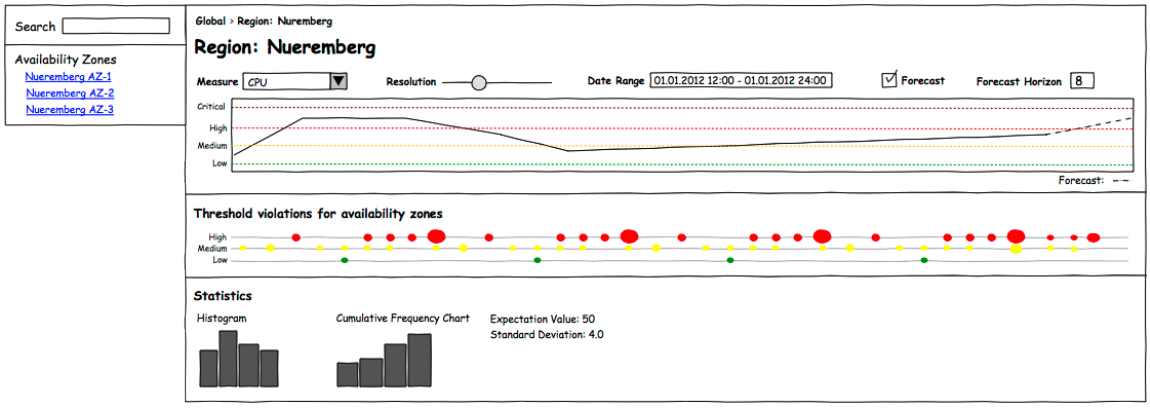


Figure 11.7: Region view mockup

11.3.4. Instance view mockup

Since an instance does not have any child items, the instance view misses this section, as well as the threshold violation chart (figure 11.8). However, an instance has a few parent items which it belongs to. Besides the parent items that are shown in the breadcrumbs navigation, an instance also belongs to a customer, a contract and a node.

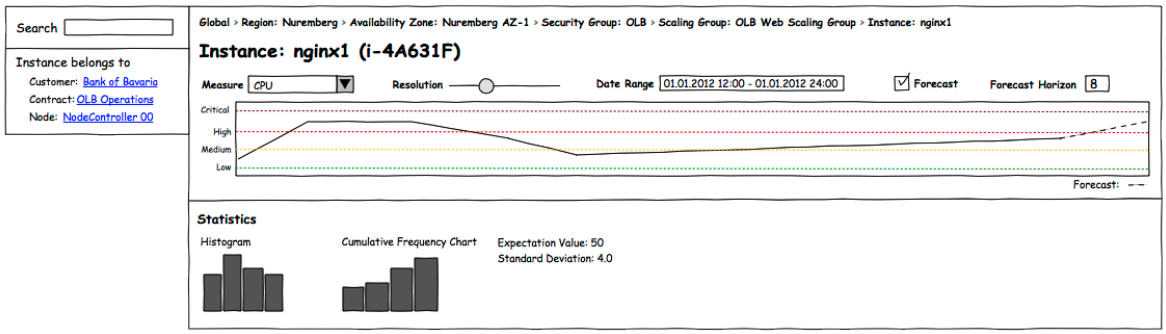


Figure 11.8: Instance view mockup

11.4. Interface design

The interface design is a part of the object design activity in the software engineering process and has the aim to specify interfaces of objects and services in detail [4]. The purpose of the *IaaS-Adapter* subsystem is to get an abstraction from the domain model of a specific IaaS provider. Therefore, the adapter pattern is used [8]. To realize the requirements that were elicited in section 10, this adapter subsystem has to be able to list the availability zones of an IaaS provider

in a certain region etc. This functionality was encapsulated into an interface, named *DomainService*. Another service that the *IaaSAdapter* subsystem has to provide is a service that queries thresholds that exist for a given metric and a server or a group of servers. This service is named *ThresholdResolverService*.

The *ThresholdViolation* subsystem is concerned about finding threshold violations in time series data. The only service within this subsystem is the *ThresholdViolationDetectionService*.

The *TimeSeries* subsystem is the biggest subsystem and contains three services: *MonitoringDataService,* that is concerned with getting TimeSeries objects for an availability zone etc. and getting the metrics that are available for servers. The *TimeSeriesService* builds TimeSeries objects from the data coming from the Storage subsystem and the *PredictionService* is responsible for calculating forecasts for time series.

An overview of the designed interfaces can be found in figure E.1.

Part IV.

Implementation and Evaluation

12. Implementation

The implementation chapter describes the different paradigms and third-party components that are used to implement the system. At the beginning, the Google Web Toolkit is explained in section 12.1 which is used for the client side code of the system, followed by an explanation of the model view presenter pattern (section 12.1.1) and an description of the "Activities & Places" framework (section 12.2). Subsequently, the technique for client-server communication is described (section 12.3), as well as the tool which is used to inject implementations on the client and server side (section 12.4). Section 12.5 describes the library that is used for drawing the charts. The implementation of the forecast algorithm is explained in section 12.6. At the end of this chapter, the package structure of the system (section 12.7) and the status of the implementation is described (section 12.8). A list of important third-party components of the system's setup can be found in the appendix chapter F.

12.1. Google Web Toolkit

The Google Web Toolkit (GWT) is a framework for implementing the client side code of websites. The big difference to other frameworks is that the client side code can be written in Java, so it does not need to be written in JavaScript. This helps to identify type errors already at compile time since Java is statically typed. Furthermore, there exist much more development-support and testing tools for Java than for JavaScript. With GWT it is possible to use tools like JUnit or TestNG for writing unit tests for the client side code as it is common for Java applications. GWT dynamically generates optimized JavaScript code during development time and updates the generated JavaScript code with a web page refresh (given that the GWT plugin is installed).

Since the client side Java code is executed within a browser, some parts of the code will need access to a DOM-tree which complicates the unit tests of these classes. GWT faces this problem with a class called *GWTTestCase* that starts a real browser instance in the background and runs the unit tests against it. However, it is advisable to minimize the number of test cases that depend on a *GWTTestCase* because the browser invocation is rather slow. Using the Model-View-Presenter pattern (MVP) is one technique to minimize these test cases because it decouples the "view part", that is responsible for displaying buttons etc. in a browser, from the model and presenter. This has the effect that only the "view-part" will need access to a proper DOM-tree and thus require a GWTTestCase. All the other unit test cases for models and presenters do not need it.

In the following, the MVP pattern itself as well as the implementation of it which is based on GWT's "Activities & Places" framework is explained.

12.1.1. Model View Presenter pattern

The MVP pattern consists of three conceptual parts:

- a **model** part that encloses data objects,
- a **view** part that is repsonsible for laying out UI elements, like headings, tables and buttons, but without a notion of the model.
- and as third part, a **presenter** that communicates with the model to retrieve and update data. It also sets data onto the view and reacts on events that are fired by the view.

The relationships between the different components of the MVP pattern are illustrated in figure 12.1.

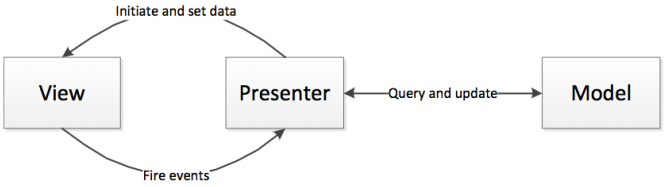


Figure 12.1: Model View Presenter pattern

12.2. Activities & Places framework

For the implementation of the MVP pattern in the project of this thesis, Google's "Activities & Places" framework is used. This framework manages the browser history by encapsulating the state of the UI of a website in *Place* objects. These objects can be serialized to strings which are automatically appended to the URL of the site. When navigating to an URL, the "Activities & Places" framework parses the parameters passed along with the URL using place tokenizer objects that are defined in the place history mapper. This results in a place object which is then used by the activity mapper to determine the activity which has to be started for this place (figure 12.2).

Figure 12.2: Processing flow for URLs in the "Activities & Places" framework

The concept of an activity is comparable to the presenter part of MVP and thus, the "Activities & Places" framework helps to implement the MVP pattern. An activity declares an interface which it expects from views. It reads data from the model and sets data onto the view. It also registers click and event handlers for events that occur within the view (e.g. the value of an input field has changed) (figure 12.3). The view part is split-up into two chunks: a declarative XML template and a Java part. They are automatically bound during compilation time with GWT's UIBinder feature.

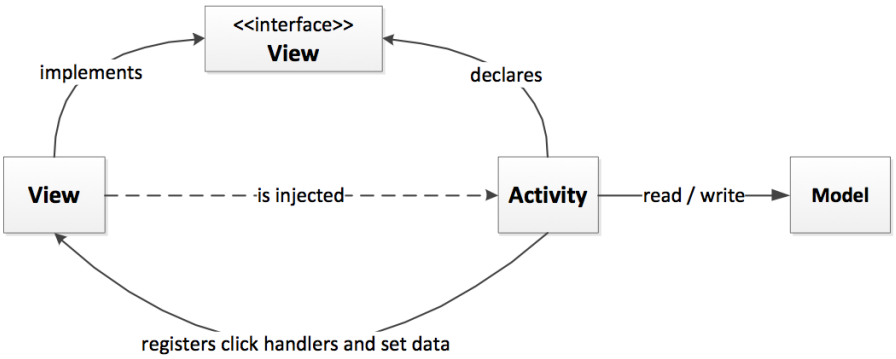


Figure 12.3: MVP implementation

12.3. Client-server communication: RPC with command pattern

For the client-server communication the implementation uses the *gwt-dispatch* package which uses the command pattern on top of GWT's RPC functionality. RPC allows a system to invoke a method that is available on a remote server.

The command pattern consists of three conceptual parts: an *action* that encapsulates parameters of a method call, a *handler* that executes an action and a *result* that is returned by the handler after executing the action.

The reasons for using the command pattern are diverse:

- the failure handling for asynchronous calls to a server can be centralized
- several commands can be requested at once using a batch mechanism
- commands can be cached, so that identical commands that are fired almost at the same time do not fire two AJAX calls to a server
- commands can be logged or queued to establish a history back functionality

Ray Ryan proposed in his Google IO (2009) talk "GWT App Architecture Best Practices" to use the command pattern on top of GWT RPC. This resulted in a project by David Peterson called *gwt-dispatch* that consists of base classes that implement the proposition Ray Ryan gave.

For every item of the sitemap (figure 11.3) an activity exists. Hence, there is an activity for the GlobalView named GlobalMonitoringActivity, an activity for the RegionView named RegionMonitoringActivity and so on.

The flow of actions that are needed to fetch data from the server are depicted in figure 12.4. When the InstanceMonitoringActivity requests a time series object for an instance it creates an action object (GetInstanceTimeSeriesAction) that contains the instance identifier and the attributes to which the time series should comply. After that, the activity registers this action together with a callback at the dispatcher. The dispatcher sends an AJAX request to the server side

of the application. The server then tries to figure out which handler to invoke for the requested action. In the example case, this is the handler GetInstanceTimeSeriesHandler because this class is associated with the requested action. The handler then queries the services of the system and builds a result object (GetInstanceTimeSeriesResult) that is returned to the dispatcher. The dispatcher receives the result object and invokes the callback that was registered by the activity at the beginning.

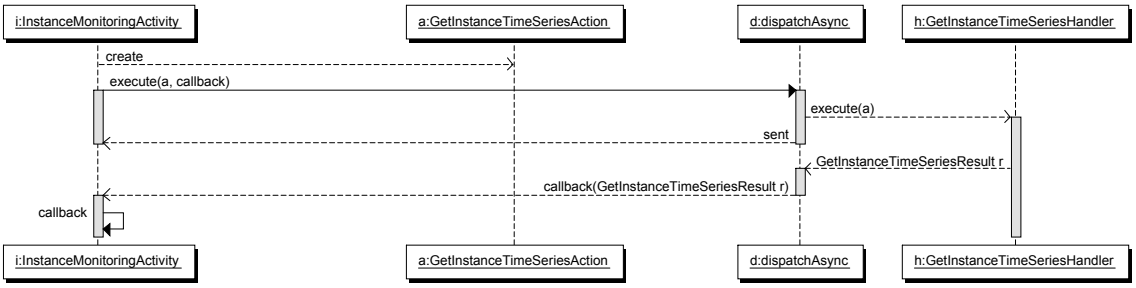


Figure 12.4: UML sequence diagram for the client server communication

12.4. Injecting interface implementations

For automatically injecting concrete interface implementations into the system, two tools are used that implement the dependency injection pattern. The dependency injection pattern aims to generalize the concept of the factory pattern and helps to maximize the decoupling of application code from specific implementations.

On the server side the dependency injection tool "Google Guice"[1] is used. The configuration class for it can be found under *capacityplanning.gui.server.guice.CapacityPlanningModule.java*. It binds the different each service interface of the system to a concrete implementation and also specifies which handler (e.g. GetGlobalTimeSeriesHandler) has to be invoked on the arrival of an action (e.g. GetGlobalTimeSeriesAction) request. This class contains also the switch to bind either a dummy implementation (DomainServiceDummyImpl) or the implementation which communicates with the FI-TS infrastructure (NimbusDomainServiceImpl) to the DomainService interface.

For the client side the tool "Google Gin"[2] is used. The configuration for Gin is located in the class *nimbus.management.gui.client.gin.ManagementGinjector.java* which is an extend version of FI-TS Gin configuration. Gin is used here to inject implementations of activities.

[1] http://code.google.com/p/google-guice/, last accessed on June 15, 2012
[2] http://code.google.com/p/google-gin/, last accessed on June 15, 2012

12.5. Visualizing time series data

The GWT plugin of Google's Visualization API[3] is used for visualizing the time series data. This GWT plugin is used with small modifications of the code base. For the visualization of the time series the LineCharts are used. For viewing histograms and the cumulative frequency charts ColumnCharts are used and for the threshold violation chart ScatterCharts are used.

12.6. Calculating autoregressive models

The computation of the autoregressive models is done by using the Yule-Walker equations with the algorithm of Paul Bourke [2]. Bourke's code has been translated to Java and the part which solves linear equations has been delegated to the Apache Commons Math package. The implementation uses the strategy pattern for the forecast algorithm so that it is easy to extend the system with other forecast algorithms [8].

12.7. Java package structure

The package structure of the project can be seen in figure 12.5. The upper most package *capacityplanning* contains four sub-packages: *timeseries*, *threshold*, *provider*, *gui*. Each of these sub-packages consists of a *dto* package that contains the data transfer objects (DTOs) that are interchanged with other subsystems and a *service* package which is split up into an *api* package for interfaces and an *impl* package for implementations of these interfaces. The *gui* package was split up into a *server* part that contains the configuration for the dependency injection framework Guice (*gui.server.guice*) and a package for *handler* objects of the command pattern that is used for RPC. Another sub-package of the GUI is the *client* package that contains the activities and places resulting from the usage of the Google Activity & Places framework. The parts of code that are shared between the server and the client are located in the sub-package *gui.shared*. Within this package exist the action and result objects that wrap the content that is passed around with RPC.

12.8. Implementation status

The implementation realizes all but two of the functional requirements that were described in section 10.1. The detection of threshold violations (FR04) involves only a dummy implementation of the ThresholdResolverService, that returns a static set of thresholds for every request, instead of querying an API of an IaaS provider. Furthermore, the search capability was not finished completely. At the current stage, the search is performed statically on the client side.

The nonfunctional requirements are all met by the implementation. With the use of the adapter pattern the part of the system that fetches information from an IaaS provider got independent from a specific IaaS provider (NR01). The calculation of forecasts is scalable with the number

[3] http://code.google.com/p/gwt-google-apis/wiki/VisualizationGettingStarted, last accessed on June 15, 2012

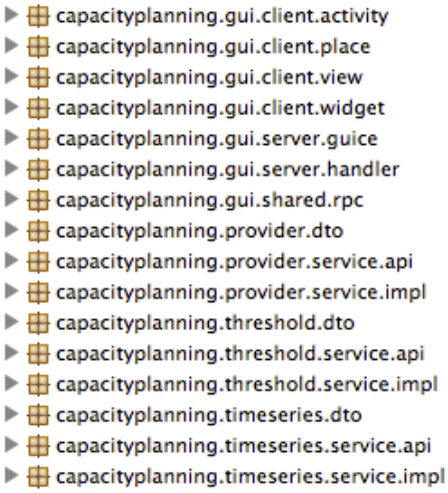


Figure 12.5: Package structure of implementation

of servers an IaaS provider operates because the calculation is done on a request basis (NR02). The forecast algorithm is exchangeable because the implementation uses the strategy pattern (NR03). The requirement for forecasts to take less than a second is ensured by an unit test (NR04). The system is independent from a specific database because it uses an data access object (DAO: *nimbus.cassandra.dao.MonitoringDAO*) for requesting data from the database (NR05).

13. Evaluation

This chapter sheds light on how the implementation of the system can support capacity planners in the questions that have been posed in chapter 1 (see section 13.1). Furthermore, it evaluates the forecast accuracy that was gained with the system as it was deployed at FI-TS (section 13.2). Finally, problems that are encountered with the current implementation are described in section 13.3.

For screenshots of the instance and the availability zone view, please refer to the appendix chapter D.

13.1. How the system supports capacity planners

For the question, when to order new servers, the global view and the region view are of major interest for capacity planners. The capacity planners get a basis for decision-making in this question with the visualization of aggregated performance measurements over time and forecasts for this data.

At current stage, the system does not give support to capacity planners in the question of whether there are any software bottlenecks existing in the systems of an enterprise. This results from the fact that the measuring component of FI-TS currently does not measure application metrics and so the system could not be evaluated with this type of data.

The question when a data center needs how many servers is only indirectly addressed by the system because the system estimates forecasts for the resource utilization but not for the actual number of servers. Nevertheless, the system is a good starting point for capacity planners in this question. They can possibly use the system's data to extrapolate the number of servers needed.

For the question, when to shutdown a server the NodeView and the InstanceView is of interest of a capacity planner. If they show constantly a very low resource utilization it might make sense to shutdown this node or instance.

With the threshold violation overview chart, capacity planners can drill down to the cause of a capacity bottleneck more easily and can thus react quicker to these events.

13.2. Forecast accuracy

To evaluate the accuracy of the forecasts, the system was deployed to the IaaS environment of FI-TS. The system was used there to predict the CPU utilization of a security group that contains web servers of a customer of FI-TS. Thereby, the forecasting horizon was set to five steps ahead.

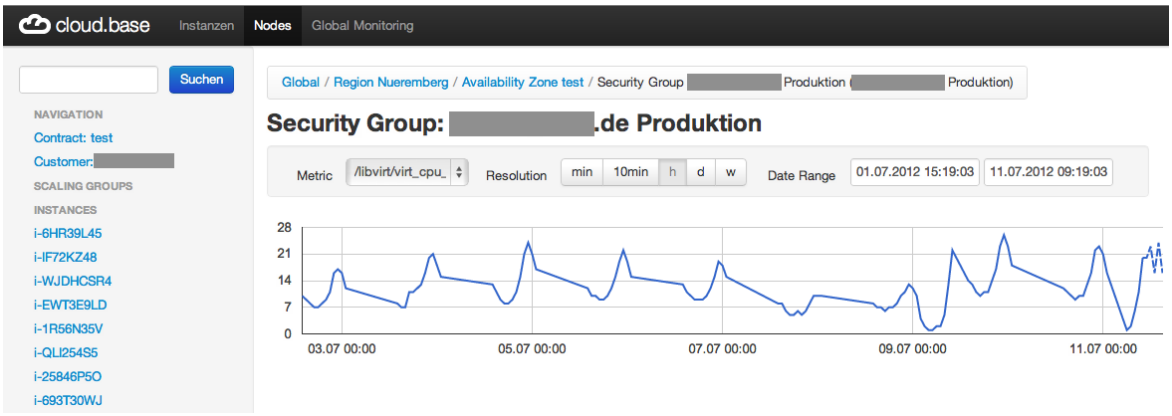


Figure 13.1: Screenshot of security group view with one hour resolution and forecasts

Figure 13.1 shows the screenshot which was taken from this setup. The dashed part of the time series shown there represents the estimated values. The time values and the estimated data values of table 13.1 were taken from this screenshot. The data for column three of table 13.1 was extracted from figure 13.2, which shows the system five hours after the first screenshot was taken. The absolute error measures the absolute difference between the estimated values and the observed values. For calculating the relative error, the absolute error is divided by the respective observed value.

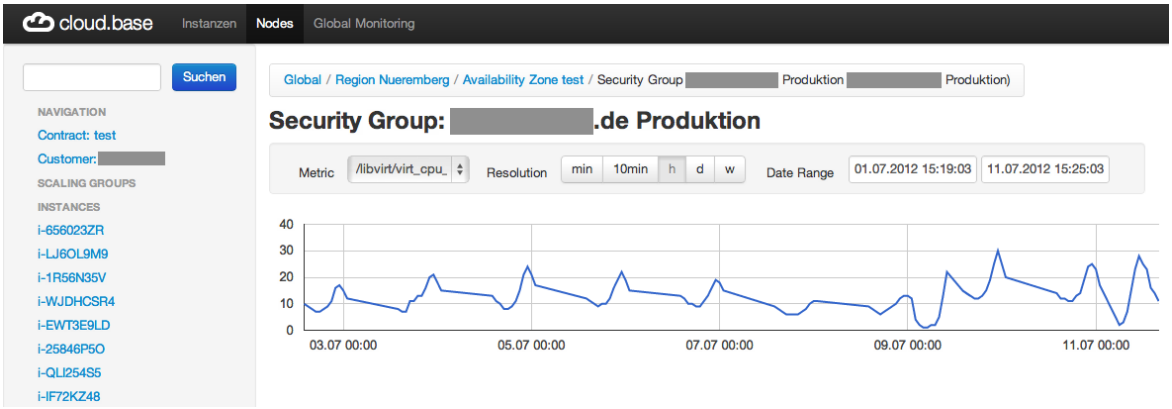


Figure 13.2: Screenshot of security group view with one hour resolution

As accuracy measure, the mean percentage error (MPE) is used. It is defined as the average over the relative errors [24]. For the dataset of table 13.1 the MPE is 24.5%, which means that the forecasts were 24.5% afar from the observed data on average. This shows that users of the system should not rely blindly on the forecasts of the system but rather take the forecast values as rough estimates. However, it is difficult to decide already right now from this small dataset whether the forecast algorithm performs well or bad in general. Therefore, an extended dataset needs to be analyzed but this is beyond the scope of this thesis.

Time	Estimated data	Observed data	Absolute error	Relative error
July 12 11:00	20%	25%	5%	20%
July 12 12:00	23%	25%	2%	8%
July 12 13:00	16%	23%	7%	30%
July 12 14:00	24%	16%	8%	50%
July 12 15:00	16%	14%	2%	14%

Table 13.1.: Forecast evaluation dataset

13.3. Problems encountered with the implementation

During the implementation of the system a problem was encountered: the SecurityGroupView might show incomplete data because the system only queries the database for historical data of those instances that are currently part of the security group but it misses those which have been temporarily in the security group. Same holds true for the ScalingGroupView, the CustomerView, the ContractView and the NodeView. The GlobalView, the RegionView and the AvailabilityZoneView have similar problems but for these views the effect is less serious because they show an aggregation of the resource utilization of nodes, which are less likely to change their location than instances. This problem has not been resolved because it would have needed bigger changes within the API of FI-TS.

14. Extension points

There are several opportunities to extend the system that was developed during this thesis.

Integrate other forecasting algorithms
The prediction subsystem can be extended with further forecasting techniques besides autoregression (Google Prediction API). This is a feasible extension because the system uses a strategy pattern for the forecasting algorithm. It is supposable that several forecasting algorithms exist in the system and track their prediction accuracy with a measure like the mean squared error for every metric of a server and to decide on the previous history of prediction accuracy which algorithm to use. Currently, the system recomputes the forecasts for every HTTP request. With the command pattern that was implemented for the client server communication it is very feasible to implement a caching mechanism for the requests that contain forecasts. Besides that, it is also possible to distribute the calculation of forecasts to a dedicated server.

Correlation analysis
Another extension to the threshold violation subsystem would be to consider correlations between threshold violations to supervise which violations depend other violations. This will help to understand overload and failure situations of applications and systems.

Integrate a sophisticated threshold violation detection algorithm
At current stage the system has only limited capabilities for finding threshold violations because the violation levels have sharp boundaries and its decisions depend on single point measurements; here a system with fuzzy boundaries that uses a bigger set of measurements for its decisions would be advantageous.

Integrate Service Level Agreement monitoring
From a business perspective, an integration of the different service level agreements (SLA) would be reasonable. With its forecasting mechanism the system will be able to throw warnings if a violation of a SLA approaches.

Extensions at FI-TS
FI-TS can extend the system to feed their auto scaling service with forecast values. It is also conceivable to integrate the system with a ticketing system. The tickets could be displayed in the time series chart, so users can discover the consequences of status changes of tickets.

Standalone version
Another conceivable extension is a complete standalone version of the system, that is not tied anymore to FI-TS related systems or infrastructure. Therefore, an own authentication system has to be implemented and some of the FI-TS GWT utilities, like the wrapper around the RPC-

Requests, have to be replaced. Also the GWT views have to be adapted. Two big parts of the system have to be implemented for this: the collection of performance metrics and the system to store them in Cassandra.

15. Conclusion

This thesis aimed at supporting capacity planners of IaaS providers in their decisions with forecasts on performance data and the detection of threshold violations within the data. Therefore, the whole process from measuring, collecting and storing monitoring data is described. The stored data is then used to build statistical models for estimating forecasts and for detecting threshold violations. A navigation concept using mockups is proposed that simplifies the navigation through the monitoring data. The mockups were realized as a web dashboard based on GWT that visualizes the monitoring data as well as the forecasts and the occuring threshold violations.

Moreover, the benefits of the system for capacity planners were described: ranging from faster reaction time resulting from quicker problem identification, to the decision support for buying additional servers. Furthermore, conceivable extensions for the system are proposed, like the integration of other forecasting algorithms besides Autoregression and the integration of service level monitoring and the extension for considering cross-correlations between threshold violations.

For an evaluation of the forecast accuracy, the system was deployed to the IaaS environment of FI-TS. A dataset was collected and it revealed that the forecasts of the system differed on average by 24.5% from the actually observed data. However, it is difficult to say whether the system performs in general good or bad because the dataset was rather small. Future work could focus this issue by collecting and analyzing a larger dataset.

A problem that was encountered during the implementation is that some of the views of the system, like the SecurityGroupView, might show incomplete data due to the temporal fluctuations of instances within a SecurityGroup. This problem could not be resolved because this would require to extend the API of FI-TS.

Since the implementation's code is to a large extend independent from FI-TS, it is also conceivable for other IaaS providers to use it. FI-TS will use the forecasting functionality of the system and embed it into their internal GUI which manages IaaS resources. The chart visualization functionality, however, will not be used by FI-TS since the terms of service for the Google Visualization API are not acceptable for them.

Appendix

A. Terminology

Performance monitoring data: are resource utilization measurements of servers.

Metric: is a type of a measurement; e.g.: CPU usage, I/O throughput, RAM usage.

Datapoint: is a single measurement of a metric for a server.

Instance / Virtual machine: is a virtual server that runs on a physical server with only little knowledge of the real hardware.

Hypervisor: is a software abstraction layer on top of a server's real hardware and manages the resource allocation between the different virtual machines which are running on it.

B. Abbreviations

Business
ITIL: IT Infrastructure Library
SLA: Service Level Agreement

Cloud Computing
EC2: Elastic Compute Cloud
GCE: Google Compute Cloud
IaaS: Infrastructure as a Service
PaaS: Platform as a Service
SaaS: Software as a Service

Software architecture
MVP: Model View Presenter
RPC: Remote Procedure Call
GWT: Google Web Toolkit
DI: Dependency Injection
REST: Representational State Transfer
AJAX: Asynchronous JavaScript and XML
(G)UI: (Graphical) User Interface
DAO: Data Access Object
DTO: Data Transfer Object

Cassandra
CF: Column Family
SCF: Super Column Family

Others
FI-TS: Finanz Informatik Technologie Service GmbH
ISO: International Organization for Standardization
UPS: Uninterruptible Power Supply
ARIMA: Autoregressive Integrated Moving Averages
AR: Autoregression
MPE: Mean percentage error

C. Mockups

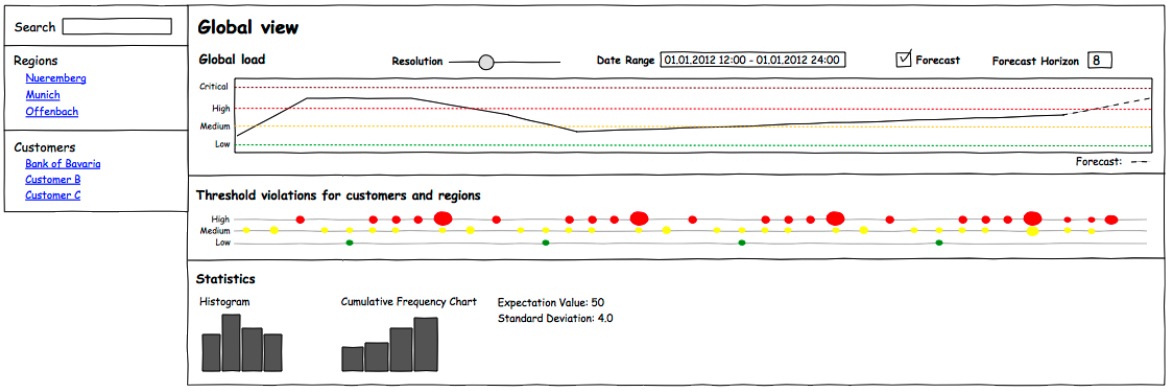


Figure C.1: Global view mockup

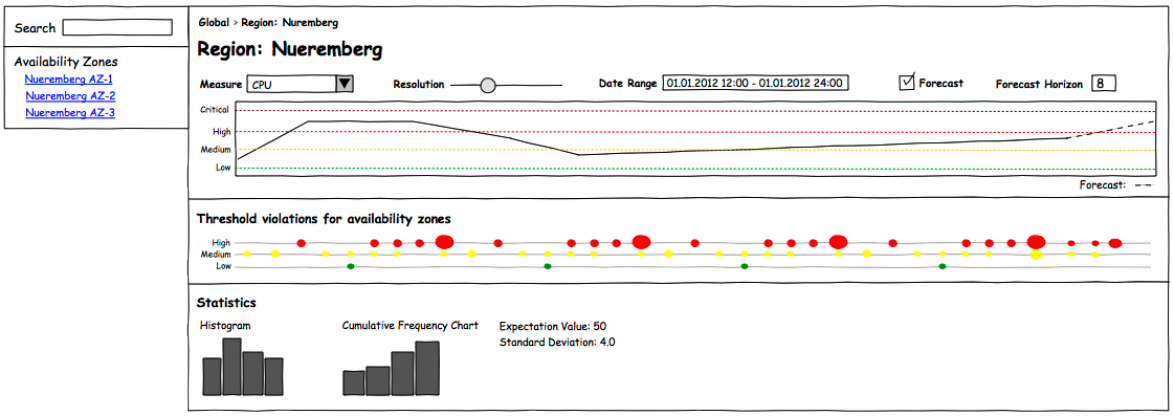


Figure C.2: Region view mockup

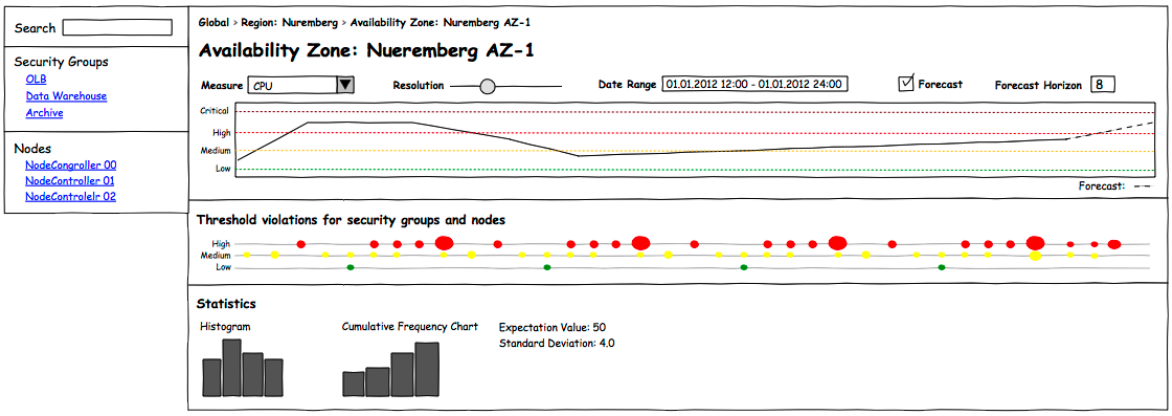


Figure C.3: Availability zone view mockup

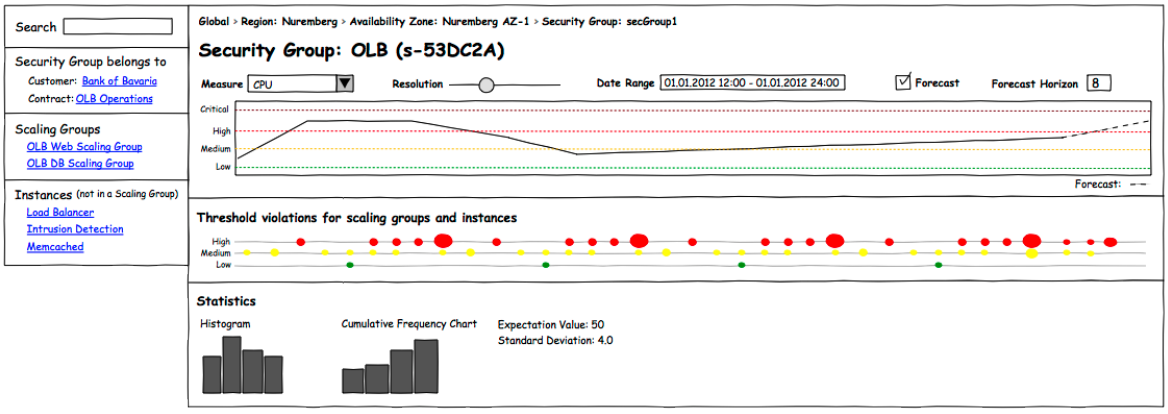


Figure C.4: Security group view mockup

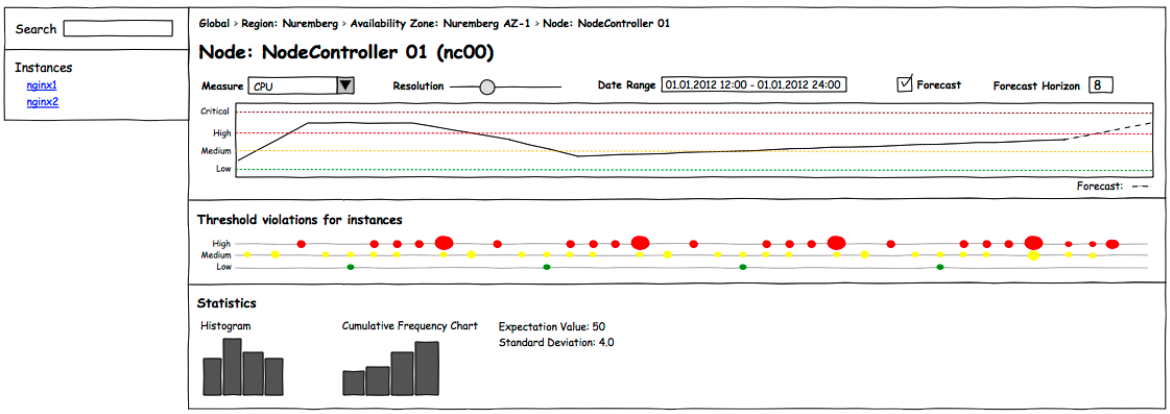


Figure C.5: Node view mockup

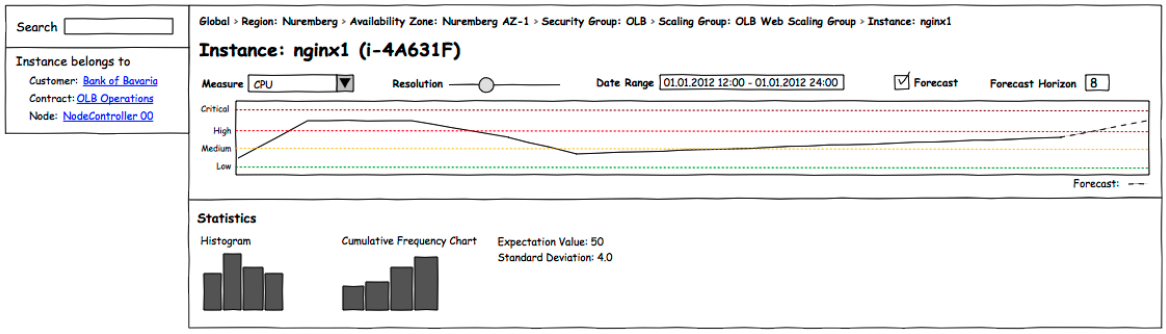


Figure C.6: Instance view mockup

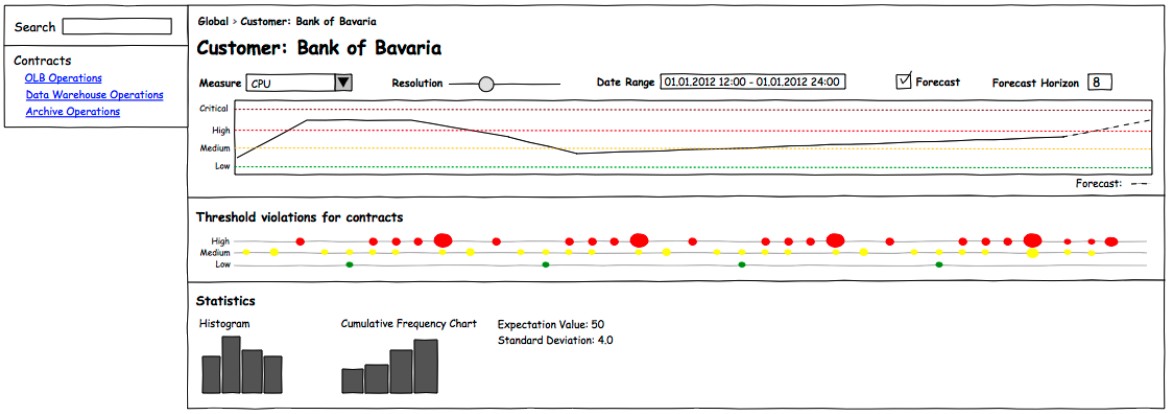


Figure C.7: Customer view mockup

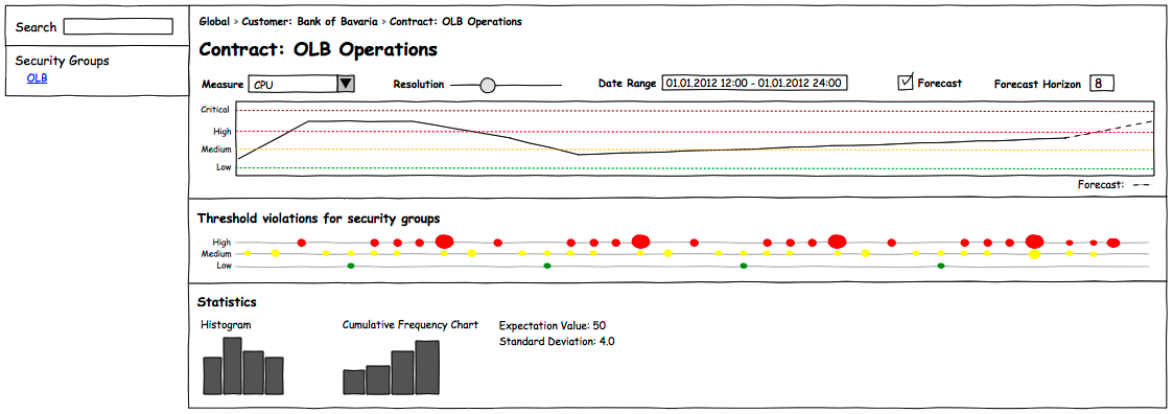


Figure C.8: Contract view mockup

D. Screenshots

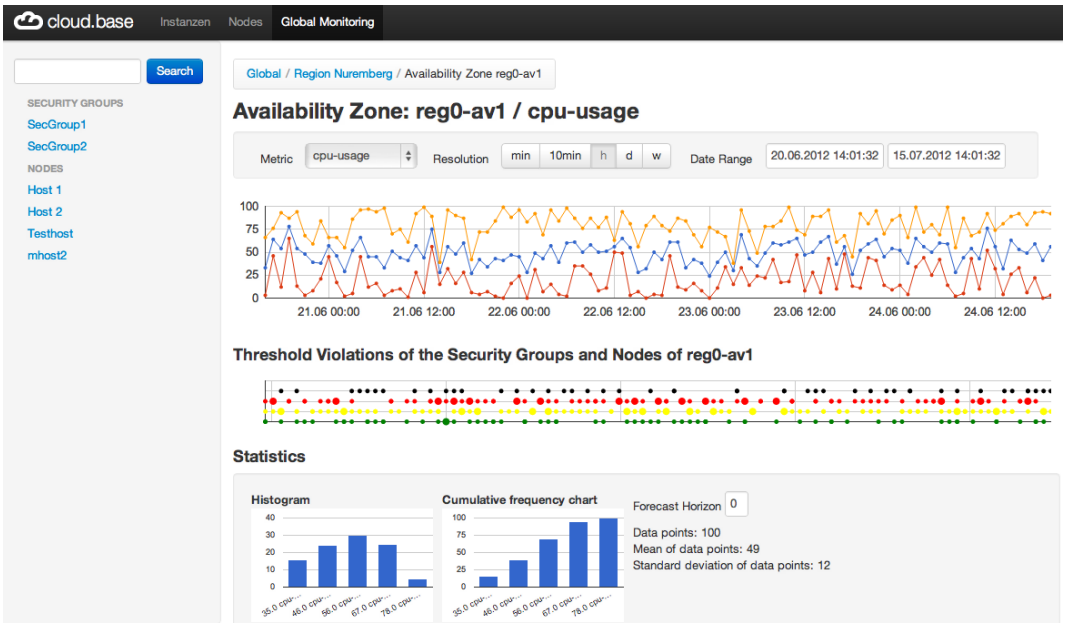


Figure D.1: Screenshot of the availability zone view

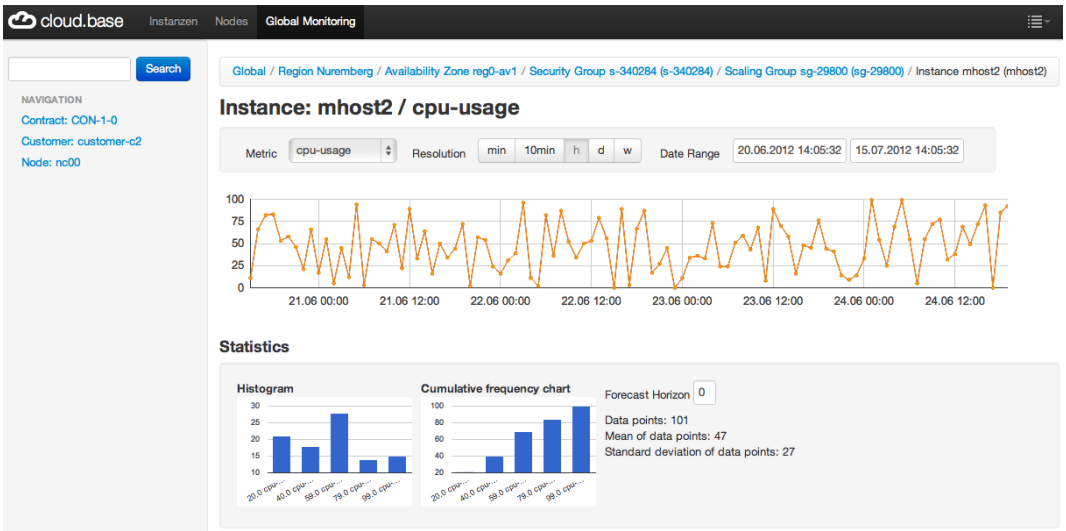


Figure D.2: Screenshot of the instance view

E. Interfaces

Figure E.1: Interfaces specified during interface design

F. Setup

- Java 6
- Google GWT 2.4: https://developers.google.com/web-toolkit/
- GWT Dispatch 1.2.0: http://code.google.com/p/gwt-dispatch/
- Google Guice 3.0: http://code.google.com/p/google-guice/
- Google Guava 10.0.1: http://code.google.com/p/guava-libraries/
- Apache Commons Math 2.2: http://commons.apache.org/math/
- Apache Cassandra 1.0.6: http://cassandra.apache.org/
- Scale 7 - Pelops 1.3: https://github.com/s7/scale7-pelops
- Twitter Bootstrap: http://twitter.github.com/bootstrap/
- GWT Bootstrap 2.0.3.0: https://github.com/gwtbootstrap/gwt-bootstrap
- GWT Visualization API 1.2.1[1]
- Google Gin 1.5.0: http://code.google.com/p/google-gin/

A local Cassandra installation is needed to start the application of the thesis. It is advisable to import the sample data which was appended to the thesis' CD to this local installation. Therefore, the contents of the folder *cassandraData* has to be copied to */var/lib/cassandra/data/*. After that, Cassandra can be started from the command line by navigating to the *bin* directory its home and initiating this command: *./cassandra -f*. The sample data contain random monitoring data starting at June 20, 2012. So it is advisable to use this date as start date within the time series attributes section.

To run the war file which can be found on the CD, a Java servlet container is needed; preferably jetty [2]. The part that was developed during the thesis is accessible with the link "Global Monitoring" within the main navigation.

[1] http://code.google.com/p/gwt-google-apis/wiki/VisualizationGettingStarted
[2] http://jetty.codehaus.org/jetty/

G. CD

Contents:

- Eclipse workspace of the implementation part (folder: **eclipseWorkspace**):
  - **management-gui**: contains the extensions the were made to the Management GUI of FI-TS; contains the component that was developed in this thesis.
  - **cassandra-dao**: contains the DAO for accessing Cassandra (impl. by FI-TS).
  - **visualization-api**: the adapted version of Google's Visualization API for GWT.
- PDF version of the thesis
- LaTeX source code and figures
- Literature
- Cassandra sample data (folder: **cassandraData**)

Bibliography

[1] M. BEIMS, *IT-Service Management in der Praxis mit ITIL© 3*, Hanser Fachbuchverlag, 3rd ed., 2012.

[2] P. BOURKE, *AutoRegression implementation in C*, `http://paulbourke.net/miscellaneous/ar/`, 1998. last accessed on July 9, 2012.

[3] P. J. BROCKWELL and R. A. DAVIS, *Introduction to Time Series and Forecasting*, Springer, 2nd ed., 3 2002.

[4] B. BRÜGGE and A. H. DUTOIT, *Object-Oriented Software Engineering Using UML, Patterns and Java*, Pearson Education Inc., 2007.

[5] CASSANDRA PROJECT, *Data Model*, `http://wiki.apache.org/cassandra/DataModel`. last accessed on July 9, 2012.

[6] G. DECANDIA, D. HASTORUN, M. JAMPANI, G. KAKULAPATI, A. LAKSHMAN, A. PILCHIN, S. SIVASUBRAMANIAN, P. VOSSHALL, and W. VOGELS, *Dynamo: amazon's highly available key-value store*, SIGOPS Oper. Syst. Rev., 41(6), Oct. 2007, pp. 205–220.

[7] P. A. DINDA, *Design, Implementation, and Performance of an Extensible Toolkit for Resource Prediction in Distributed Systems*, IEEE Trans. Parallel Distrib. Syst., 17(2), Feb. 2006, pp. 160–173.

[8] E. FREEMAN, E. FREEMAN, B. BATES, K. SIERRA, and E. ROBSON, *Head First Design Patterns*, O'Reilly Media, 1st ed., 2004.

[9] S. GILBERT and N. LYNCH, *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*, SIGACT News, 33(2), June 2002, pp. 51–59.

[10] D. GMACH, J. ROLIA, L. CHERKASOVA, and A. KEMPER, *Capacity Management and Demand Prediction for Next Generation Data Centers*, in ICWS 2007, IEEE International Conference on Web Services, July 2007, pp. 43 –50.

[11] D. GMACH, J. ROLIA, L. CHERKASOVA, and A. KEMPER, *Workload Analysis and Demand Prediction of Enterprise Data Center Applications*, in IISWC 2007, Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization, IEEE Computer Society, 2007, pp. 171–180.

[12] F. I. T. S. GMBH, *Company Profile*, `http://www.f-i-ts.de/medien/downloads/FI-TS_FI-TS_FI-TS-Abspaenne-2012.pdf`. last accessed on July 9, 2012.

[13] N. J. GUNTHER, *Guerilla Capacity Planning*, Springer-Verlag Berlin Heidelberg, 2010.

[14] T. HAERDER and A. REUTER, *Principles of transaction-oriented database recovery*, ACM Computing Surveys, 15(4), Dec. 1983, pp. 287–317.

[15] E. HEWITT, *Cassandra: The Definitive Guide*, O'Reilly Media, 2010.

[16] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, *ISO/IEC 27001:2005*, `http://www.iso.org/iso/catalogue_detail?csnumber=42103`. last accessed on July 9, 2012.

[17] J. LIANG, K. NAHRSTEDT, and Y. ZHOU, *Adaptive multi-resource prediction in distributed resource sharing environment*, in CCGrid 2004, IEEE International Symposium on Cluster Computing and the Grid, April 2004, pp. 293 – 300.

[18] J. PREVOST, K. NAGOTHU, B. KELLEY, and M. JAMSHIDI, *Prediction of cloud data center networks loads using stochastic and neural models*, in SoSE 2011, 6th International Conference on System of Systems Engineering, June 2011, pp. 276 –281.

[19] R. H. SHUMWAY and D. S. STOFFER, *Time Series Analysis and Its Applications*, Springer Science+Business Media, LLC, 3rd ed., 2011.

[20] M. STANLEY, B. PAPER, A. HOLT, K. WEISS, K. HUBERTY, and N. ROZOF, *Cloud Computing Takes Off Market Set to Boom as Migration Accelerates*, 2011. last accessed on July 9, 2012.

[21] THE NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, *Definition of Cloud Computing*, `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`, 2011. last accessed on July 9, 2012.

[22] J. TIDWELL, *Designing Interfaces*, O'Reilly Media, 2nd ed., 2011.

[23] G. UPTON and I. COOK, *A Dictionary of Statistics (Oxford Paperback Reference)*, Oxford University Press, USA, 2nd ed., 2008.

[24] WIKIPEDIA, *Mean percentage error - Wikipedia, The Free Encyclopedia*, `http://en.wikipedia.org/w/index.php?title=Mean_percentage_error&oldid=424121838`, 2012. last accessed on July 14, 2012.

[25] R. WOLSKI, *Dynamically forecasting network performance using the Network Weather Service*, Cluster Computing, 1, 1998, pp. 119–132.

[26] R. WOLSKI, N. T. SPRING, and J. HAYES, *The network weather service: a distributed resource performance forecasting service for metacomputing*, Future Generation Computer Systems, 15(5–6), 1999, pp. 757 – 768.