Karlsruhe Institute of Technology

Modeling and Experimental Analysis of Virtualized Storage Performance using IBM System z as Example

Diploma Thesis of

Dominik Bruhn

At the Department of Informatics
Institute for Program Structures
and Data Organization (IPD)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. Dr. Walter F. Tichy
Advisor:	Dipl.-Inform. Qais Noorshams
Second advisor:	Dr.-Ing. Samuel Kounev

13th February – 12th August

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 2012-08-12

.....

(**Dominik Bruhn**)

Zusammenfassung

Im Zuge der zunehmenden Anforderungen an die Verfügbarkeit, Skalierbarkeit und Effizienz unter Performanzgarantien von Software– und Hardwaresystemen hat sich Virtualisierung als eine Technologie durchgesetzt, die diese Anforderungen erfüllen kann. Durch diese Entwicklung gerät auch die Performanz von persistentem Speicher vermehrt in das Blickfeld: Wenn immer mehr Last auf einer Maschine gebündelt wird, steigen gleichzeitig die Anforderungen an die persistente Speicherhardware und deren Performanz. Beschleunigt wird diese Entwicklung durch den zunehmenden Bedarf an schnellen Antwortzeiten für Anfragen auf große Speichermengen.

Im Umfeld von virtualisierten Maschinen treten viele Fragestellungen im Zusammenhang mit persistentem Speicher auf: Für Systemadministratoren ist bespielsweise die Frage wichtig, in welchem Maße Änderungen der Systemeinstellungen Auswirkungen auf die Speicherperformanz des Gesamtsystems haben. Für Anwendungsentwickler sind degegen Abschätzungen über die Antwortzeiten von Anfragen an das Speichersystem hilfreich, um die Gesamtperformanz ihrer Anwendungen abschätzen zu können. Diese Fragen können nur schwierig im laufenden Betrieb beantwortet werden. Falls kein mit dem Produktivsystem vergleichbares Testsystem zur Evaluation zur Verfügung steht, ist eine Abschätzung der Speicherperformanz ohne Änderungen am Produktivsystem wünschenswert. Diese Probleme können mit statistischen Modellen gelöst werden, für deren Erstellung einmalig Messdaten auf dem entsprechenden System gesammelt werden müssen. Nachdem die Modelle erzeugt wurden, liefern sie Vorhersagen der Performanz bzw. der Antwortzeiten von Speicheranfragen ohne weiteren physischen Zugriff auf das System und ohne dessen Veränderung.

Vorhandene Arbeiten betrachten entweder die Gesamtperformanz des Systems in ihren Untersuchungen und Vorhersagen oder fokussieren sich auf die Untersuchung der Performanz von persistenten Speichersystemen auf der Betriebssystemebene ohne Berücksichtigung des Dateisystems. Diese Betrachtungen können nur indirekt helfen, die Anwendungsperformanz abzuschätzen.

In der vorliegenden Diplomarbeit wird ein Ansatz basierend auf statistischen Regressionsmodellen verfolgt. Um diese erstellen zu können, wird zuerst der Einfluss der verschiedenen Parameter analysiert und quantifiziert, wobei die hierfür benötigten Daten durch systematische Messungen der persistenten Speicherperformanz gewonnen werden. Mit dem Wissen über den Einfluss der Parameter, werden aus den gemessenen Daten Regressionsmodelle erstellt.

Diese Regressionsmodelle werden detailliert untersucht und ausgewertet. Ihre Qualität wird anhand von mehreren Metriken beurteilt und eingeordnet. Zusätzlich werden verschiedene Regressionstechniken benutzt, analysiert und verglichen. Es erfolgt außerdem eine Einschätzung, auf welchem Wege die Modelle verbessert werden können, unter anderem, indem die Regressionstechniken angepasst oder andere Messdaten verwendet werden.

Der Ansatz zeigt gute Ergebnisse: Es werden Performanzdaten des persistenten Speichers in virtuellen Maschinen gesammelt, die auf einer IBM System z Maschine ausgeführt werden, wobei eine IBM DS8700 als Speichersystem benutzt wird. Basierend auf diesen Messdaten werden Regressionsmodelle erstellt, die die Antwortzeiten von persistenten Speicheranfragen mit einem relativen Fehler von 3.8% vorhersagen. Die Unterschiede der verschiedenen Regressionstechniken werden durch die Qualität der Vorhersagen der Modelle deutlich. Zudem unterscheiden sich die Techniken im Zeitbedarf für die Modellerstellung und die Vorhersage, in der Komplexität der Algorithmen und in der Interpretierbarkeit der Modelle.

Abstract

In recent years, the increasing demand for resource scalability and resource efficiency as well as demands for greener IT led to the widespread use of virtualization technology. Server consolidation through virtualization provides a solution to optimize data center operating costs, administration overhead and resource flexibility. Having increasingly virtualized environments, virtualized storage performance requires more and more attention: If more load is concentrated on a single machine, the requirements for performance of the storage hardware also increases. This development is accelerated by a growing demand for fast response times of storage requests working on huge data sets.

In the field of virtualization, many questions concerning storage and its performance arise: For system administrators, the question of the influence of changes of system settings on the storage performance is important. For application developers, an estimate of the response time of storage requests is helpful to assess the overall application performance. These questions can not be answered easily on running machines: If there is no evaluation system available, which is comparable to the production system, an estimate of the performance of storage systems without changing the production system is helpful. A possible solution for these problems are prediction models: They are created using measurements which have to be gathered on the system once. After their creation, the models can then be used to answer the questions by providing a prediction for the performance and the response time of storage requests. They provide a prediction without the need for further physical access on the system and without its modification.

Even though the need for practical storage performance prediction approaches is high, there are few existing approaches that thoroughly analyze and evaluate virtualized storage systems. These approaches either focus on the overall system performance without including the storage system in detail or focus on the evaluation, analysis and prediction of the performance of storage systems at the operating system layer. The latter approach does not include the file system and can therefore not be used directly to predict the application performance.

This thesis presents a systematic performance analysis and evaluation approach for I/O–intensive applications in virtualized environment. First, an in–depth analysis and quantification of the parameters, which influence the storage performance, is conducted. The data which is needed for this process is obtained from systematic measurements. Second, statistical regression models are created based on the systematic measurements, using the knowledge on the influencing parameters. Next, these regression models are analyzed in detail to evaluate their quality. In a trade–off analysis, different regression techniques are compared and checked for their applicability on the prediction of storage performance. Finally, an assessment of how the regression techniques can be enhanced is presented.

The approach shows good results: The storage performance measurements of virtual machines are systematically collected. The virtual machines are executed on an IBM System z and the storage is provided by an IBM DS8700 system storage. The regression models which are created during this thesis can predict the response time of a storage request with a relative error of as low as 3.8%. The differences between the regression techniques are shown in detail: These differences include the quality of the models, the time required for the creation of the models and the prediction of samples, and the complexity and interpretability of the models.

Contents

1. Introduction

In recent years, the increasing demand for flexibility and cost efficiency and the focus on topics like green IT and cloud computing led to the widespread use of virtualization. Server consolidation through virtualization provides a solution to the requirements of low costs, flexibility, administration and scalability.

Having increasingly virtualized environments, virtualized storage performance requires more and more attention: This results from the fact that the aggregation of multiple systems on one host machine leads to an increasing demand for storage performance. Another reason for the growing request for storage performance is the demand of today's applications to work on huge amounts of data and reply to requests in short time. This is especially true for web applications and online services which work on massive amounts of data which can be searched and edited by millions of users at the same time. Another example are search engines and other services which need to handle a massive amount of data in a reasonable short time.

Virtualized storage is often neglected by the current software performance engineering approaches [NKR12]. There exists little information, understanding and knowledge about the influence of the different system and workload parameters and their interaction on the storage performance. Most current approaches either take a black–box approach for the virtualized storage system (e.g. cf. [AAH+03]) or adopt an approach which involves sophisticated and full–blown simulations (e.g. cf. [HBR+10]).

If storage performance is analyzed and examined, this is typically done taking the systems or the operating systems view (e.g. cf. [CKK11, WAA+04]). This means that the performance of the storage is examined as seen by the operating system without taking the file system into consideration. The view of the application on storage is different due to the mechanisms and policies introduced in the operating system itself, especially at the file system layer. This view of the application on storage performance and the importance of storage performance at the application layer is often neglected. As this is the view of the application developer and in the end also the view of the user of the software, it is important to consider this viewpoint on storage performance.

It is often desirable to predict the performance of storage requests. Both, an application developer and a system administrator typically need to predict the storage performance after having made changes to the system or their application. This is difficult if the host system and the storage system are not available for testing, either because they are in use as production system and cannot be spared for evaluation or because the systems are

not present at all, for example when evaluating the acquisition of a new storage system. The influence of a setting or parameter change on the system is also difficult to evaluate because it is often not possible to change the system as it is used in production and an interruption of the service should be avoided. Another difficulty is the estimation of the storage performance after a software change. The information if, for example, an increase in read requests has an influence on the overall software performance can typically only be obtained on the actual system which is often impossible because to the facts specified above. When operating virtual machines another use of storage performance models is the prediction of changes to the system performance when changing the virtual machine setup: An example usage are two virtual machines which are running on two separate hosts and the operator wants to know if an aggregation on a single host degenerates the performance of the virtual machines. If this decision should be made automatically by the management software, storage performance models are essential.

Statistical storage performance modeling helps in all these cases: It provides a statistical regression model which can be used to predict the storage performance without the need for access to the machine. Nevertheless, the performance data which is used to create the regression model must be obtained on the actual system. Based on these one–time measurements, the regression models can be generated. After this, they can be used at anytime to predict samples, even those not benchmarked, without any further access to the system.

When looking at statistical storage performance regression models, it is unclear how well these models perform and how usable their results are. As there exists a huge number of regression techniques, the question arises which of these techniques is suitable for modeling storage performance. It is unclear whether simple regression techniques, like linear regression, suffice, or more sophisticated regression techniques, like MARS [Fri91], CART [BFSO84], or M5 [Qui92], are needed for good results. Potentially, it might not be possible to predict storage performance at all using regression models and other solutions must be found. Additionally, the question arises if and how these regression techniques can be tailored to the prediction of storage performance.

Regression models can be helpful in two ways: First, they can be used to simply predict the storage performance. In this case, the internals of the models and their contents are not important and the focus mainly lies on the quality of the predictions. Second, the regression models can also be used to get insight to the system which was modeled. This involves the detailed analysis of the models which makes interpretability an important requirement. The decision for the goals of the modeling has influence on the choice of the regression techniques as the form and the characteristics of the regression techniques are different.

This thesis presents a systematic performance analysis and evaluation approach to I/O intensive applications in virtualized environments. In a first step, this thesis contains an in–depth statistical analysis and quantification of the parameters which influence the storage performance. The analysis is based on systematic measurements gathered on an IBM System z and an IBM DS8700 used as storage system. Using these measurements and the results from the parameter analysis, statistical regression models are created in a second step. These models are evaluated and analyzed for their quality in terms of generalization abilities, interpolation and extrapolation. Next, in a trade–off analysis, different regression techniques are compared and checked for their applicability on the prediction of storage performance. Finally, an assessment of how the regression techniques can be enhanced is presented. As explained above, the systematic measurements are gathered on an IBM System z and an IBM DS8700 storage system. These two real world systems are state–of–the–art virtualization technology. Applying, testing and evaluating the approach on a

machine like the IBM System z also assures that the process can also be applied on other real world scenarios. This thesis is based on the work previously done by Noorshams et al. [NKR12]. They use a quantitative approach to identify which parameters influence the storage performance on a virtualized system which is reused in this thesis.

1.1. Contribution

The contribution of this thesis is fourfold:

1. A statistical evaluation of storage performance influencing parameters using systematic measurements in a real world environment with state–of–the–art technology.
2. A systematic creation and evaluation of regression models for the prediction of storage performance using the results of systematic measurements.
3. An in–depth evaluation, analysis and comparison of regression techniques valid for storage performance prediction.
4. A fully automated approach for the measurements, for the evaluation and the analysis of the parameters, of the regression models, and of the regression techniques.

1.2. Outline

This thesis is structured as follows:

- *Chapter 2* contains related work. It is categorized into three groups: The first group contains publications which have a related approach to system performance modeling without a focus on storage performance. The second section includes papers which have a specific focus on storage performance and the third section contains papers which are not directly related to storage performance prediction but are related because of their work on regression techniques and modeling per se.
- In *Chapter 3*, the technical foundations are laid. This includes the explanation of the system under test and the storage hardware. Additionally, the parameters which can have an influence on this system are discussed and the benchmark used in this thesis is presented.
- *Chapter 4* contains the statistical and mathematical foundations which are needed in the later sections. This includes, after some first basic definitions, an in–depth explanation of the four regression techniques which are used later in this thesis to model the storage performance. Additionally, this chapter contains a description how regression techniques and their results can be compared in a statistically correct way.
- *Chapter 5* describes the actual setup which was used for the benchmarks, the analysis and the modeling. First, the parameters which are analyzed and their ranges are defined. Later in this chapter, an introduction to the *Storage Benchmark Harness* and the *Analysis Library* is included. These two tools were written for this thesis and are designed to automate the benchmarking, analysis and modeling process. The chapter closes with the Goal/Question/Metric plan which explains in–depth which questions are analyzed in the next two chapters.
- *Chapter 6* contains those research questions which are not related to the modeling but focus on the analysis of the benchmark results and the influencing parameters instead.
- *Chapter 7* focuses on the regression models which can be built from the collected results. It contains two sections: The first section analyzes how well the regression techniques can be applied to the data and the second section compares the different regression techniques with respect to storage performance modeling.

- *Chapter 8* closes this thesis with a summary and an outlook on future work.

2. Related Work

This section presents the related research papers. For a better overview, the papers have been categorized into three different sections: The first section contains those papers which focus on the overall system performance. The second section presents the papers which have a closer focus on storage performance. The third section lists papers which have a related approach but do focus on neither overall system performance nor storage performance.

2.1. System Performance Modeling

All papers in this section do not explicitly focus on storage performance. Instead they analyze and model the performance of a whole system.

Kundu et al. [KRDZ10] gather performance metrics of a whole system. They control the CPU, memory, disk bandwidth and network bandwidth usage limits. Afterward, they benchmark four different applications with varied values for the resource limits. Each of these four applications has its own output variable which is measured when the limits are in effect. They feature a CPU intensive, a memory intensive and a disk intensive application together with an overall system testing application. They then use their four limiting parameters as independent variables and the application dependent output as dependent variable for multiple regression models. They test several variations of the linear regression, including quadratic terms and interactions. Additionally they use artificial neural network models as an alternative modeling technique. Later the results of the linear regression are compared to the artificial neural network models. The models are automatically refined using an iterative model training. Although the authors use XEN to limit the resources of the virtual machines they do not focus about virtualization. As result of their papers, the authors show that using artificial neural network models provides significantly better results than using linear regression models. They were able to predict the outputs of the benchmark applications with a median modeling error below 6.65%. While including the storage performance in their models, they do not include any workload parameters but focus on the system limits.

Benevenuto et al. [BFS+06] decide to use simple queuing models to predict the performance overhead of a migration from a dedicated machine to a virtual machine. They use three different benchmarks and collect their output to predict this overhead. The authors use a simple mathematical formula with constant factors similar to a linear model. The results

from their benchmarks are used to compute the coefficients in the formula. They do not focus on storage and storage performance at all and even choose not to model the time required for the disk accesses.

Huber et al. [HvQHK11] run a more general approach: First, they conducted various experiments on two different virtualization platforms and on a native machine. Afterward, they calculated the virtualization overhead of some components like network, memory and disks. Later, they propose a model which helps to estimate the performance of virtualized machines. Their prediction model is based on linear regression. The paper contains a more overall view on several components while focusing on the overhead of virtualization.

In their publication, Koh et al. [KKB+07] use two virtual machines and examine which combinations of applications result in performance interference. In a second step they create a statistical model which predicts the performance of applications based on their characteristics. These characteristics include the number of blocks which are written per second and other storage and CPU related characteristics. The focus of their examinations is the performance analysis and modeling of applications not of storage requests.

2.2. Storage Performance Modeling

The contributions in this category explicitly focus on the analysis and modeling of storage performance. Their main difference lies in the parameters they include in their analysis: Some authors choose to include only workload parameters, other authors include only system configuration in their examinations. The first paper [NKR12] is the only one which models the performance at file system layer. Most papers focus on benchmarking and modeling at the block layer.

This thesis is based on the work of Noorshams et al. [NKR12]. The authors take a similar approach as this thesis for the benchmarking: They also use FFSB for the benchmarking and variate both, the workload and system parameters. They use the same IBM system as the one used for this thesis. Their main focus lies on the analysis of the performance influencing factors. Additionally they include some basic modeling approaches using linear regression models, mainly to show which parameters have a linear correlation.

Wang et al. [WAA+04] use CART models to predict storage performance. They use two separate sets of input variables: Their first set uses five different workload configuration variables as independent variables, the second set uses detailed descriptions of each request issued as independent variables. Their dependent variable is the response time of a request. They do not run any benchmark but instead rely on pre–existing traces from real world storage systems. The CART models are trained using parts of these traces. Other parts of the traces are used to validate the models. The authors compare the two CART models to two predictors which do not include the workload parameters but instead use constant values. Linear regression models are also included in the comparison. The whole work uses block layer requests and the authors do not include the file system in their models. In their results the authors show that predicting the response time from both, workload description and detailed request information, is feasible using CART models. These models outperform the linear regression and the two constant predictors.

In their paper, Ahmad et al. [AAH+03] use mathematical models for predicting the storage performance for a migration from a native machine to a virtualized environment. They use benchmark runs from the native server and the virtualized instance to conduct a model. They only varied the block size, the read/write–ratio and the ratio of requests which are random or sequential. As a result the authors do not predict the response time or performance of the storage requests. Instead only the overhead of virtualization is predicted. By also focusing on the block layer, they leave out the whole file system.

Kraft et al. [KCK+11] use queuing network models to predict I/O performance in virtualized environments. They use traces, gathered from the block layer of the virtualized Linux machine, to build a model for the device contention. In their follow–up work [CKK11, KCK+12], the same authors extend this model to predict the expected I/O performance after a consolidation of virtual machines. They use benchmark results from the individual isolated virtual machine and describe a model which calculates the performance which can be expected if the machines are consolidated. Like their previous mentioned paper, they run their benchmarks on the block layer and are therefore unable to predict application performance.

Huber et al. [HBR+10] measure the response time and throughput of storage block operations. The block size, the read/write–ratio and the number of clients is varied. Later a performance model is build from this measured data. To model the collected data they use a PCM model [BKR09]. The PCM model is validated using newly benchmarked data and an overall approach validation is done by using the PCM models to decide between two design alternatives. The authors conclude that the PCM is not perfectly suited to model storage performance. Nevertheless their approach shows how valuable storage performance prediction models are. Although the authors do not clearly state that they are not including the file system but are benchmarking at the block layer, this can be concluded from their approach specification.

The authors Anderson et al. [And01] use a similar approach as this thesis for the storage performance prediction: They first gather the raw results and then use simple modeling techniques to predict the response time or the throughput. They show how simple models can be generated and discuss why a nearest neighbor approach is not suitable for storage performance prediction. Instead they recommend an approach similar to linear regression models. Nevertheless this work stays at a conceptual level: No results and no actual data is provided.

In their paper, Lee et al. [LK93] propose a queuing model for the modeling of the utilization of a hard disk array. They calculate the constant factors for this model from the hard disk parameters and later prove that their models operate at a very low error rate when predicting the utilization. The authors focus on the low level structure of the hard disk and only model direct hardware requests. They do not include the block layer or even the file system in their considerations.

2.3. Regression Analysis and Comparison

The authors Westermann et al. [WHKF12] take a similar approach as a later chapter of this thesis: They compare different regression techniques and their performance when modeling collected data. They evaluate MARS, CART, Krigin [vBK04] and a genetic programming approach [Fab11]. They use synthetic data generated from a predefined formula to train these models and then compare how well the regression techniques approximated the formula. For the validation of their approach, they test their models on an optimization of the Java Virtual Machine. While their approach comparing different regression techniques resembles the one in this thesis, their focus is not on storage performance.

Faber [Fab11] presents an approach how to compare genetic programming to MARS. The focus lies on genetic programming, which was not used in this thesis. Nevertheless, the approach of using a GQM plan to compare different regression models and regression techniques resembles the one in this thesis.

Courtois and Woodside [CW00] examine how Multivariate Regression Splines (MARS) models can be used to predict properties of computer systems. They focus on the CPU

time needed to send and receive TCP packets. In their paper, the authors also show what the advantages of using MARS are and how it can be applied to other fields of study.

In their paper, Kim et al. [KLSC07] compare different regression tree algorithms by looking at their interpretability and the quality of their predictions. They include the CART and the M5 algorithm which are also used in this thesis. They compare the models using many different data sets from different fields of study, including computer science, medicine, and sports.

There are various publications which compare regression techniques in other fields of study: Ture et al. [TKKO05] use them to predict the risk of a heart disease. In their work, they compare various regression techniques in detail, including CART and MARS which are also used in this thesis. Moisen and Frescino [MF02] compare five regression techniques, including linear regression models, MARS, and CART, to predict the characteristics of forest trees. The authors Chae et al. [CHC+01] compare different regression techniques and how they can be used to predict data from the topics of health insurance. They include different regression trees, for example CART, in their comparison.

3. Technical Foundations

This chapter gives insight into the technical constructs used in the thesis. It describes the system which is used as the system under test and the software which is run on the system under test.

3.1. IBM System z

The *IBM System z* is a series of mainframe computers designed and manufactured by IBM. The systems are designed for high availability. Therefore the whole system is built using spare components which support hot fail over to ensure zero downtime. The fail–over mechanisms are implemented in the system firmware and are independent from the operating system.

A typical IBM System z machine contains up to 80 processors (so–called *processing units* or *PU*). These processors can support different features: For example the *Central Processor* (*CP*) can be used as a generic processor, whereas an *Integrated Facility for Linux* processor (*IFL*) can only be used to run Linux virtual machines. The customer can, to some extend, configure what processors should be built into his machine. It supports up to 1520GB of main memory also depending on the actual configuration of the system.

The System z is a heavily virtualized system [Mei08]. Its virtualization features are older than the virtualization technology on X86 machines. It features a complete virtualization of nearly all of its hardware: CPU, memory and network are the most prominent virtualized components. The system architecture supports two levels of virtualization: On the lower level, the PR/SM hypervisor allows the partitioning of the systems hardware into so–called LPARs. In one of these LPARs either an operating system or the second layer hypervisor z/VM can be run. Both levels of virtualization are supported by the hardware architecture and are therefore quite fast. The main difference is that main memory can only be statically assign to LPARs. They do not use any memory virtualization. If memory virtualization is needed, z/VM can be used.

The *IBM DS8700* is used as storage system for the IBM System z for this thesis. The storage system can save a huge amount of data (up to 2048TB [DBC$^+$10]) on its hard disks. The DS8700 features a sophisticated caching system which saves slow disks access for often used data. It consists of two redundant management systems (so–called *Processor Complexes*) which control the disks, provide management facilities and contain the memory

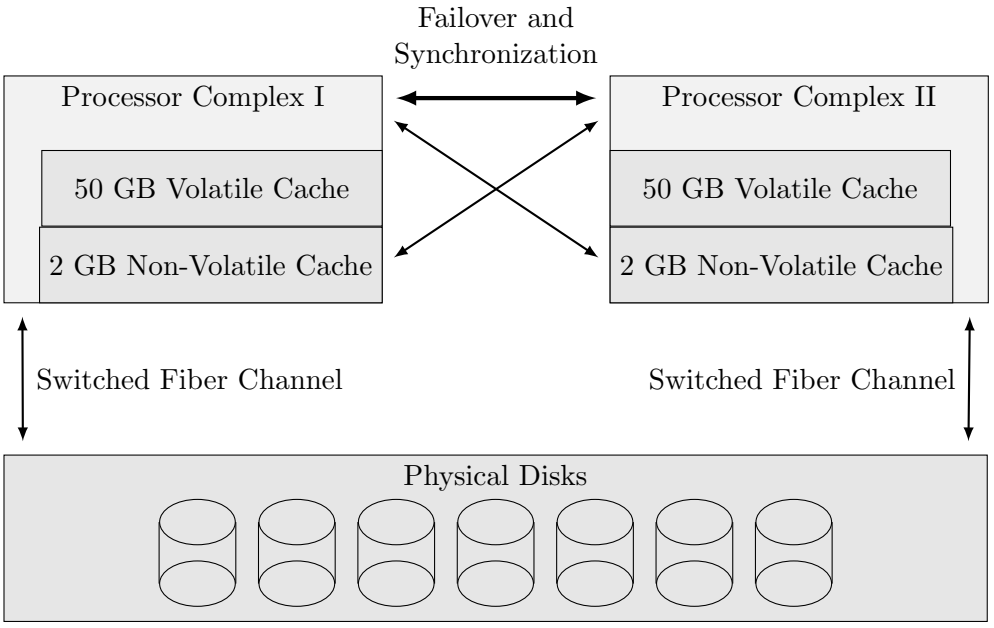


Figure 3.1.: Architecture of the IBM DS8700.

which is used for the caches. Each of the Processor Complexes contains two caches: A non–volatile cache (2 GB), where write requests are saved and a big volatile cache (50 GB) where read requests are cached. The processor complexes are statically assigned to the underlying storage. This means that one of the processor complexes is responsible for a specific volume. There is no load balancing and only if a failure occurs, the other processor complex handles the requests. A write requests is accepted by one of the Processor Complexes and instantly written to the non–volatile cache of the other Processor Complex. This is done to save the request if a failure of the processor complex occurs. If no failure occurs, the write request is deleted from the non–volatile cache of the other processor complex after the data has been successfully written to the disk. The cached write data is only accessed if the processor complex fails before it is able to store the data permanently on the physical disks.

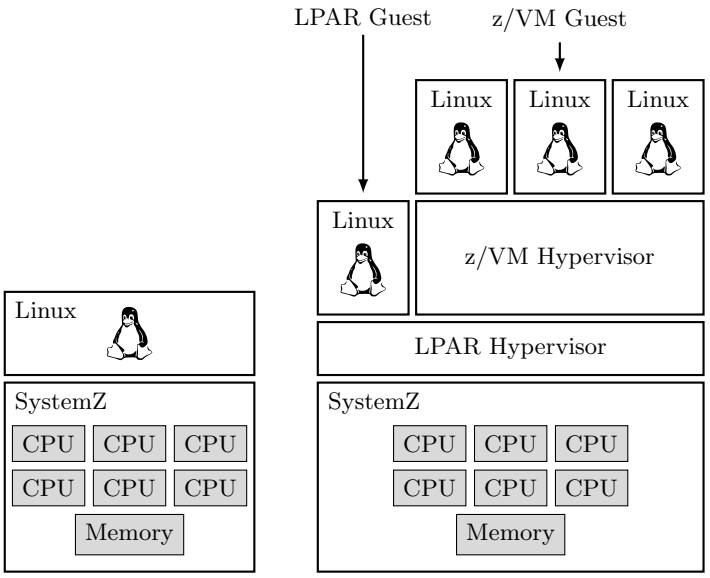
The DS8700 is virtualized which means, in this case, the abstraction from the physical drives to virtual partitions or logical volumes [DBC+10]. The DS8700 provides abstract logical volumes for the attached systems and internally distributes the data on many disks. Figure 3.1 shows the architecture of the IBM DS8700.

3.2. Linux & Linux on IBM System z

Linux is an open source operating system which can be run on a variety of machines and architectures. IBM started to support Linux on the IBM System z in 2000 [DSW03]. Typical use cases for Linux on IBM System z include the migration of Linux application from other servers to the IBM System z and the usage of software which is not available on IBM's default operating system z/OS. Additionally, the usage of Linux helps to ensure portability: Because of the huge amount of architectures supported by Linux, it is easy to port applications written on one system to another, e.g., run application written for the X86 architecture on an IBM System z.

Linux can be run in three ways on the IBM System z [DSW03]:

- *Basic or native mode*: Only one Linux instance is run and it is the only operating system on the whole machine. This resembles a Linux installation on an ordinary



(a) Basic (b) LPAR and z/VM

Figure 3.2.: Options to run Linux on IBM System z.

desktop computer. It is also called "bare metal". This mode is very uncommon configuration for machines like the IBM System z as a single Linux instance can usually not utilize the resources provided by a typical IBM System z machine.

- *Logical Partition*: The LPAR Hypervisor (also called PR/SM) partitions the IBM System z in so–called logical partitions (short LPAR). There can be at most 15 partitions. Each of these partitions can run a different operating system. For example Linux can be run in one of the partitions while z/OS is running in another one. The resources are statically assigned to the partitions and thus dynamic resource sharing is not possible. Processing units are statically assigned to the logical partitions. This setup is typically used for heavily used machines which are fully using their resources and thus need to statically assign them to the LAPRs.
- *z/VM*: To add more virtual machines to the IBM System z and to make resource sharing easier, another level of virtualization can be used: *z/VM* is an operating system which only purpose is to run as hypervisor and thus enable virtualized machines to use its resources. It is run inside an LPAR. z/VM supports hundreds of virtual machines and can so be used to share resources dynamically between virtual machines.

Figure 3.2 contains a graphical representation of these options. As the Basic mode is very uncommon on machines like the IBM System z, this option is not considered in this thesis.

An IBM System z is typically accompanied with an external storage device. These machines use an array of hard disk to store the data which is used by the IBM System z securely. In the case of this thesis the DS8700 is used for storing the data and the operating systems of the virtual machines. The hypervisors in the IBM System z can emulate a conventional SCSI device towards the virtualized Linux machines [DBC+10, PBH+08]. Therefore, the DS8700 can be accessed in the same way as any SCSI device and shows no special behavior for the Linux machines.

The virtual machines which are used for the benchmarking in this thesis run the Debian Linux distribution. The choice for Debian was made because of its free and open source nature and its ability to change the internals of the system easily. The Debian version

was fixed to 6.0.2 during this thesis as any update would change the software version and influence the benchmark results. Nevertheless, it should be noted, that Debian is not an officially supported Linux distribution by IBM.

3.3. Possible Performance Influencing Factors

The performance of a workload depends upon two major factors: The workload itself and the system configuration. Both can be varied in benchmarks. In the following sections, possible factors for both groups are listed. See Figure 3.3 for a graphical representation of the factors and their relation. During the thesis, some of the parameters are selected and are then included in the model. Other parameters are difficult or impossible to variate. This leads to the fact that some parameters cannot be varied automatically and require manual operation. For example, it is impossible to change the RAID level during the thesis as this would involve rebuilding the whole system and potentially losing all data.

During the thesis, not only the parameters but also their possible values are examined. While this was easy for some parameters (for example *Request Type*), it is difficult for others. *Request Size* for example can range from small values to very large values. The search for sensible values is one of the tasks of this thesis and is discussed later.

The following sections discuss the workload and the system parameters in detail.

3.3.1. Workload Characterization

- *Request type*: A request to a POSIX file system can be one of the following types: *Read*, *Write*, *Create*, *Append*, *Delete*. A workload can either consist of requests from only one of these types or can be a mixed workload. In the latter case, some kind of weights must be provided to specify the ratio of each of the request types. All requests except for the *Read* and *Write* operation also involve changing of the file system meta data. This means that the operations can be categorized into two groups: Those requests which modify the file system data and those requests which solely operate on the actual data of the files.
- *Request size*: The request size is composed of two parts: The first part is the block size: It specifies how much data a single operation reads from the file or writes to it. The second part is the amount of block which should be read. Typically not only a single but multiple blocks should be read. The actual request size results from the multiplication of the block size and the block count.
- *Access pattern*: A request can be either *Sequential* or *Random*. This specifies how the blocks which should be read or written are accessed. As specified above, a read request typically reads more than one block. In random access mode the benchmark randomly requests blocks from the file while in sequential mode it sequentially reads one block after the other. Sequential accesses benefit from the precaching attempts of the controller hardware especially when operating on a huge file set.
- *Workload threads*: The amount of threads which perform file system operations simultaneously has a great impact on the performance. Each of the threads continuously issues operations on the file system and therefore on the storage device. The amount of threads is sometime also called the client count because the threads can be regarded as clients to the storage system. The behavior of the threads can be configured in detail, for example the sleep time specifies how long each thread waits after it has issued a read or write request. The amount of threads can also be used to apply higher or lower load of the system.

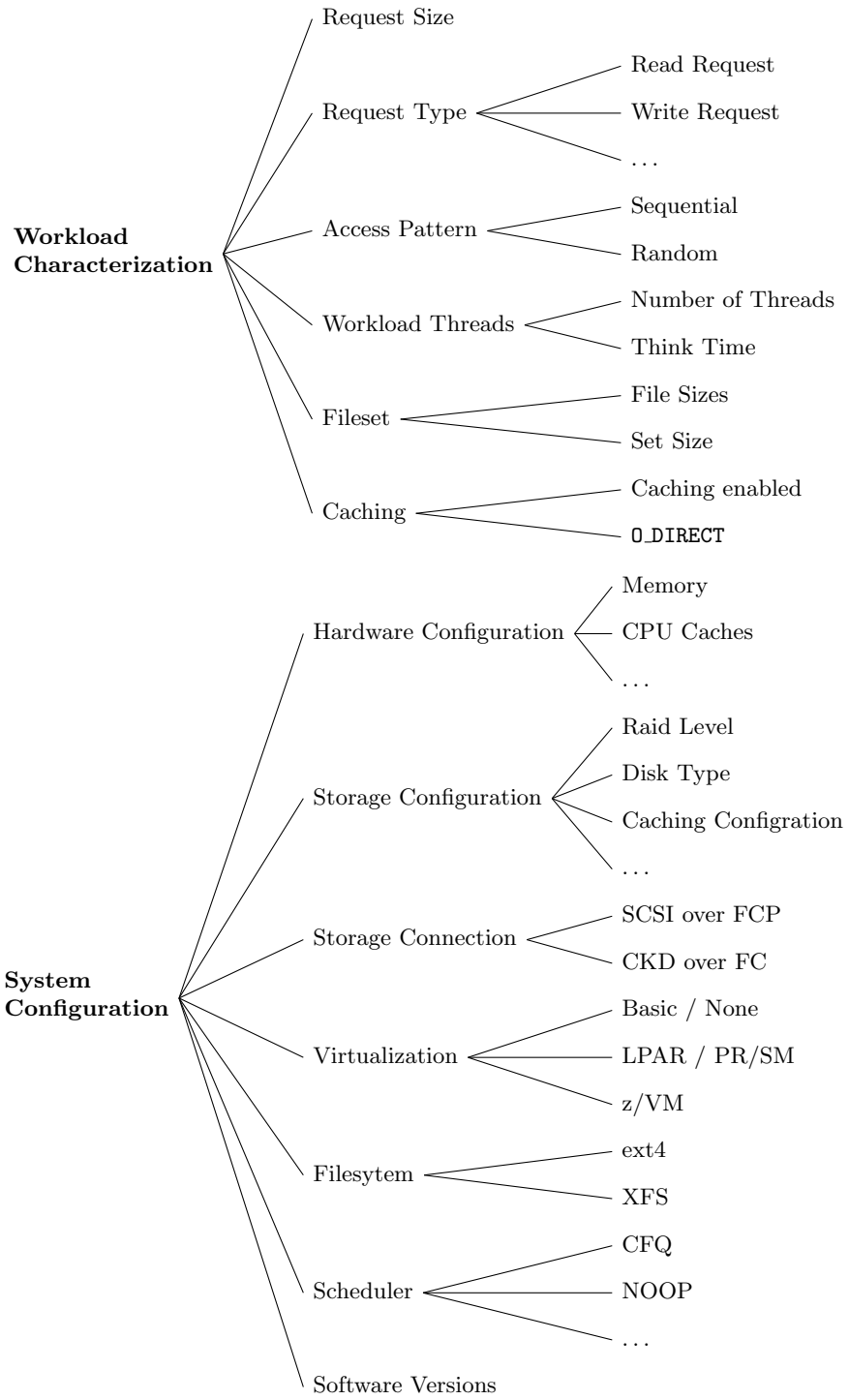


Figure 3.3.: Possible performance influencing factors (based on [NKR12]).

- *Virtualization*: The number of virtual machines on which the benchmark is run in parallel has a great influence on the response times. Therefore the amount of virtual machines involved can be regarded as an influence factor. The technology, which is used for the virtualization, cannot be regarded a workload parameter but must be seen as a system configuration instead. See below for the explanation of the virtualization technology.
- *File size*: For some requests the size of the file on which they operate on plays a crucial role. For example, the *Read* operation can be configured to read the whole file. The execution time of this operation is obviously dependent on the file size.
- *File set size*: The benchmark operates on a set of files which are generated before running the benchmark. This file set can have different sizes: Smaller file sets might fit into the cache as a whole and thus operations on such file sets can run very fast. Larger file sets do not fit in the cache and lead to hard disk accesses. As some of the requests can still be answered from the caches, large file sets have a huge variation in their response times.
- *Caching mode*: Linux caches I/O requests in the main memory for faster lookup. When using these caches, another layer in the memory hierarchy is introduced. It can be disabled by using the `O_DIRECT` flag. This disables all file system caches and directly forwards requests to the devices. However, this setting does not disable the caches in the storage system and other caches which are not handled by the operating system. Reference to Figure 3.4 for the caches in the kernel which can be disabled by this setting. For this thesis this means that even if the `O_DIRECT` flag is set, the caches in the DS8700 are still in effect.

3.3.2. System Configuration

- *Hardware configuration*: Obviously the hardware configuration has a huge impact on the I/O performance. Nevertheless, as the system can not be changed in this thesis, the whole hardware configuration must be regarded as fixed.
- *Storage configuration*: Many factors for the storage configuration like the RAID level and the RAID block size cannot be changed on a running and already setup up storage system. This makes benchmarking and modeling their influence on performance impossible during this thesis.
- *Virtualization configuration*: The IBM System z supports different virtualization modes (see Section 3.1): There might be differences between executing the virtual machines, which executes the benchmark, in an LPAR and a z/VM container. The latter adds another level of abstraction to the system.
- *Storage protocols*: The DS8700 can provide different storage protocols with different characteristics towards the virtualized Linux systems.
- *File systems*: The file systems which are often used like *ext4*, *XFS* or more experimental file systems like *Bttrfs* have different performance characteristics. Additionally each of these file systems can be configured for different workloads. For example most file systems support the configuration of the block size in which the files are stored on the hard disks.
- *Scheduler*: Current Linux versions support four different I/O schedulers: `noop`, `anticipatory`, `deadline` and `CFQ`. All of these I/O scheduler operate at the block layer and are responsible for scheduling the block requests. The following two schedulers are evaluated more closely in this thesis:

The *completely fair queuing scheduler* (short CFQ) maintains a queue for each process running on the system. Its major goal is the fair distribution of the available I/O bandwidth. The I/O requests for the process are appended to the process queue. The scheduler uses a time slicing algorithm to fairly distribute the I/O time to the processes. Each process is allowed to do the same amount of I/O. Additionally the requests are ordered to access blocks, which are stored together closely on the disk, after each other and therefore speed up the operation. This scheduler is the current default on most Linux systems.

In contrast to this complex algorithm, the *no operation scheduler* (short NOOP) is the simplest scheduler available for Linux. It simply maintains a queue for all I/O requests and works on this queue in a first–in–first–out way. No requests are reordered or time sliced. This reduces the computation time on the host. The main reason to use this scheduler is the fact that the storage system can better schedule the requests because it knows about its own internals whereas this information is unavailable to the scheduler on the host.

As the field of I/O scheduling constantly evolving, there exist a number of other schedulers: The *deadline* and the *anticipatory* scheduler are more lightweight when compared to CFQ. They focus on maintaining an upper limit for the response time of the requests. The *Fair I/O Operations Per Second Scheduler* (short FIOPS) is a new scheduler which is currently only supported by recent Linux kernel versions. The scheduler is optimized for data which is not stored on hard disks. The scheduler contains some assumptions which are true for flash based storage and tries to optimize the operations per second.

- *Software versions*: The features and optimizations included in the kernel which ships together with the Linux distribution change fast. This makes the kernel version a performance factor and must be kept in mind.

3.4. Benchmarking & FFSB

A major task during the thesis is benchmarking. In a later section, the chosen benchmark is described. The benchmark runs have to be repeated several times to get stable results. It might even be necessary to dismiss the first runs because of caching effects. For the same reason the running time of the benchmarks has to be sufficient large.

For the whole benchmarking and model generation process, the storage system is be regarded as a black–box, so no status information from the storage devices is used. This does not mean that internal information, like the size of the caches is not be included in the analysis.

The *Flexible File System Benchmark*[1] (short *FFSB*) is a cross–platform benchmark tool which operates at the file layer. This means that it issues read and write requests to files. The results are therefore collected at the same layer as any other application accesses its files at. This is different from the approach of benchmarking at block layer. At this layer, the file system does not play a role. On the one hand, the FFSB approach makes the model generation more difficult but on the other hand the results can be used to generate prediction models for the performance of storage requests for applications and not only for the operating system.

FFSB was selected because of its use in other related publications, like the research paper of Kraft et al. [KCK+11]. FFSB is freely available as open source and can be configured easily.

[1] `http://ffsb.sourceforge.net`

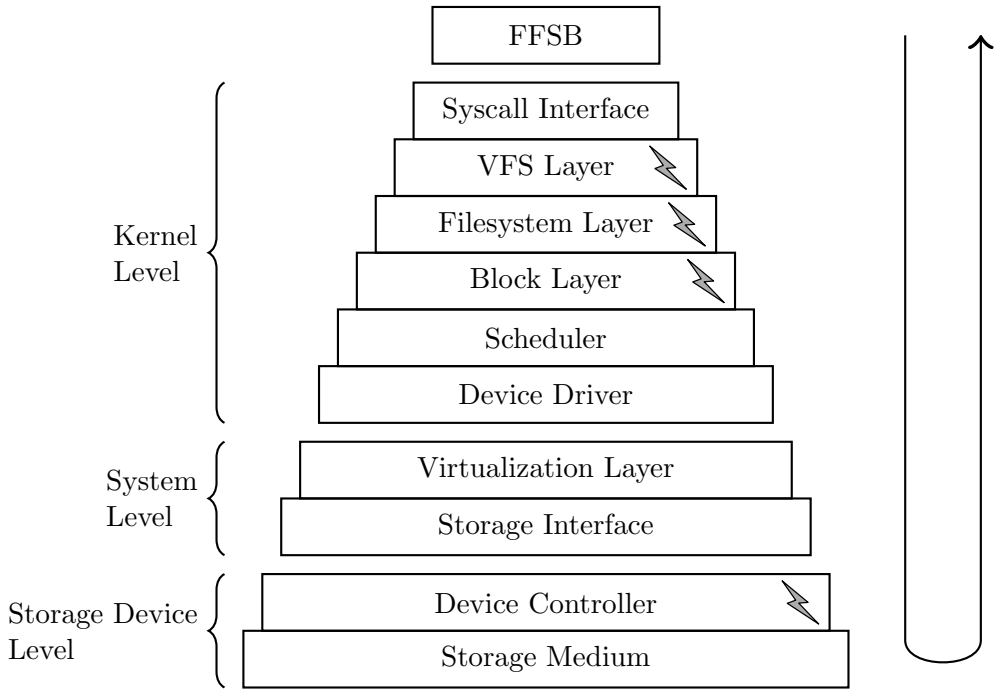


Figure 3.4.: Different layers a requests issued by the FFSB Benchmark has to go through. Layers marked with a thunderbolt contain caches which can speed up responses but also make predictions more difficult.

Figure 3.4 shows the difficulties of the benchmarking at this layer and therefore the difficulties of the whole thesis: Each requests issued by the benchmark application (FFSB) goes through a huge number of layers until the request reaches the actual storage medium. Most of the layers contain sophisticated logic and some of layers even contain caches. These are marked with thunderbolts in the figure. Those layers with caches introduce another indeterministic behavior in the request process: The response time decreases if the request can be answered from one of the caches.

FFSB is provided with a configuration file for startup. In this configuration file each of the parameters has to be set to a fixed value. The configuration files are plain text files. They specify the behavior for each of the three steps FFSB executes. In a first step it creates the file set on the hard disk. This task involves creating a predefined number of files filed with random content. The file size and the file set size can be specified for this process. The time consumed for the file set creation must not be underestimated. The creation of a 100 GB file set takes about 20 minutes on the IBM System z specified above. The second step of an FFSB run is the actual benchmarking. Multiple threads issue the requests which are specified in the configuration file. Each of the threads records the response time for each single request it has issued. The benchmarking is run until a specified duration is extended. After this, the third phases aggregates the results from all threads and calculates the minimum, maximum, median and mean for each of the request operations. Additionally FFSB can be configured to output the response times for each single operation after the benchmarking has completed. This setting allows in–depth analysis of single runs. The output of these detailed results it not ordered so it is not possible to analyze the temporal behavior of the operations. Instead the response times should be regarded as an unordered list and can therefore be used to analyze the distribution of the response times.

As most benchmarking tools, FFSB needs a stable clock source to measure the time a request needs to be completed. Although it might seem a trivial problem, measuring short

periods time with a high accuracy is difficult on computer systems. The clock source of the IBM System z has proven to be stable enough for the FFSB benchmark. Nevertheless it should be kept in mind that the clock source is a potential source of error which is difficult to trace down.

For this thesis the FFSB benchmark has been modified at several points to adapt to the needs. As FFSB is open source, modifications are easily possible. The modified source[2] was released again as open source. The modifications include the output of the single response times as explained above and modifications of FFSB to adapt to the benchmarking process in this thesis. Additional bugfixes have been incorporated in the new source code.

[2] `https://github.com/FFSB-Prime/ffsb`

4. Statistical Foundations

This chapter provides some definitions for functions and methods used in this thesis. This chapter can only serve as an overview of the topics. More detailed introductions to the topic can be found in specialized literature [HEK05, HTF11, Kou11].

4.1. Basics

This section defines some basic statistical terminology used in the next chapters and throughout the thesis.

4.1.1. Mean & Median

The *arithmetic mean* (or short *mean* or simply *average*) $\overline{y}$ of a set $Y = \{y_1, y_2, ..., y_n\}$ is calculated by using the following formula:

$$\overline{Y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

The median $\tilde{Y}$ of a sample or a set is defined as the sample for which half of the samples from the set are larger and the other half of the samples are smaller. This leads to the following definition when assuming that the samples in the set Y are ordered such that $y_1 \leq y_2 \leq ... \leq y_n$:

$$\tilde{Y} = \left\{ \begin{array}{ll} y_{((n+1)/2)} & \text{if } n \text{ is odd} \\ \frac{1}{2}(y_{(n/2)} + y_{((n+2)/2)}) & \text{if } n \text{ is even} \end{array} \right\}$$

The arithmetic mean gives an equal weight to all samples which means that outliers have a huge influence on its value [Kou11]. In contrast to that the median is not influenced by outliers.

4.1.2. Quantiles

Quantiles are a generalization of the median: The α-quantile of the set Y (expressed as $\tilde{Y}_\alpha$) divides the set in such a way that $\alpha \cdot 100\%$ of the data is smaller or equal to $\tilde{Y}_\alpha$ and $(1 - \alpha) \cdot 100\%$ of the samples is larger or equal to $\tilde{Y}_\alpha$. The definition for the quantile $\tilde{Y}_\alpha$ is, given a sorted set Y such that $y_1 \leq y_2 \leq ... \leq y_n$:

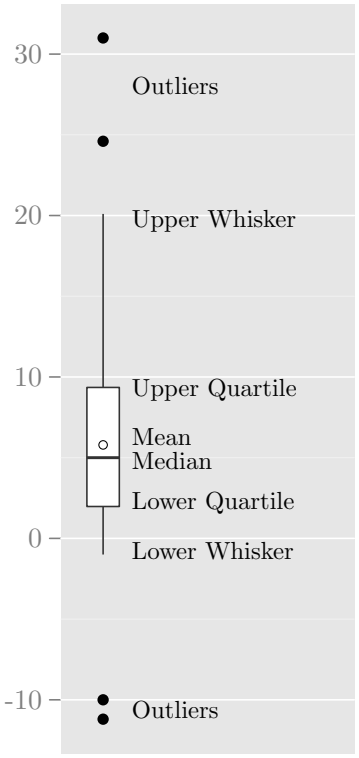


Figure 4.1.: Example boxplot including labels as explanation.

$$\tilde{Y}_\alpha = \left\{ \begin{array}{ll} y_{\lceil n\cdot\alpha \rceil} & \text{if } n\cdot\alpha \text{ is odd} \\ \frac{1}{2}(y_{n\cdot\alpha} + y_{n\cdot\alpha+1}) & \text{if } n\cdot\alpha \text{ is even} \end{array} \right\}$$

For $\alpha = 0.5$ the quantile is equivalent to the median. Other special cases are the so–called quartiles: The *upper quartile* is defined as $\tilde{Y}_{0.75}$ and the *lower quartile* is defined as $\tilde{Y}_{0.25}$. This definition directly leads to the *inter quartile range* (short *IQR*): It expresses the range between the lower and the upper quartile and therefore represents the middle 50% of a sample set. This means that the IQR is defined as $IQR = \tilde{Y}_{0.75} - \tilde{Y}_{0.25}$.

4.1.3. Boxplots

Boxplots are a way to represent and understand the distribution of the samples in a sample set. They help to quickly judge on the quality of the data by visualizing the outliers and the median. Figure 4.1 shows an example boxplot together with the meaning of the different points. The box of the boxplot contains the middle 50% of the set. This is the range between the lower and the upper quartile. The position of the whisker is not consistently defined in literature. For this thesis, the upper whisker is at $\tilde{Y}_{0.75} + 1.5 \cdot IQR$ and the lower whisker is at $\tilde{Y}_{0.25} - 1.5 \cdot IQR$. All points out of the range $[\tilde{Y}_{0.25} - 1.5 \cdot IQR, \tilde{Y}_{0.75} + 1.5 \cdot IQR]$ are called outliers and printed as dots. The line in the plot symbolizes the median and the white dot the mean. Because of a huge amount of outliers in some plots, they are not always printed. If they are omitted, this is annotated to the boxplots.

4.1.4. Cumulative Distribution Function

As explained in the previous section, boxplots can be used to show the distribution of data. Another visualization is the cumulative distribution function (short CDF) [HEK05, p. 839].

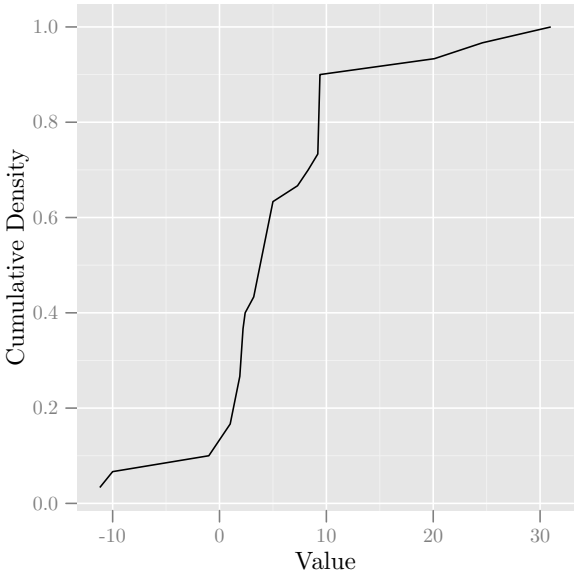


Figure 4.2.: Example cumulative distribution function using the same data as the boxplot in Figure 4.1.

An example for a CDF can be found in Figure 4.2. It shows which ratio of the samples is smaller or equal to a specific value. The ratio is depicted on the y axis and the value on the x axis. In the example, it can be seen that 40% of the samples are smaller or equal 2.5.

4.1.5. Sample Variance & Standard Deviation

The sample variance and the standard deviation are both typically used indexes of dispersion [Kou11]. The variance σ_Y^2 of a sample set $Y = \{y_1, y_2, ..., y_n\}$ is defined as:

$$\sigma_Y^2 = \frac{1}{n-1} \sum_{i=1}^{n} (y_i - \overline{Y})^2$$

It is always bigger than zero and is denoted in the squared unit of the samples. This makes the sample variance difficult to understand and leads to the standard deviation: The standard deviation σ_Y (often also written short as sd) of the sample set Y is defined as:

$$\sigma_Y = \sqrt{\sigma_Y^2} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (y_i - \overline{Y})^2}$$

The standard deviation has the same units as the samples and therefore the same units as the mean. This makes the standard deviation easier to understand. For example a big standard deviation can be caused by outliers in the data set. A smaller variance and therefore a smaller standard deviation means that the samples are closer to the mean.

4.2. Analysis of Variance (ANOVA)

The analysis of variances (or abbreviated ANOVA) is a statistical technique which can be used to compare two or more alternatives. In this thesis it is used to check if the changing of a parameters values has a significant effect. The technique accomplishes this by comparing the variation between the samples within the same group (errors) with the variation between the alternatives (effects). If the variation due to the effects proves to be

statistically bigger than the variation due to errors, then the technique concludes that the differences are a consequence of the alternative and not a consequence of the errors. This technique requires repeated measurements of the same alternatives to quantify the part of the variation which is caused by the noise of the measurements.

To better understand ANOVA, it is helpful to organize the data in a table like Table 4.1 (cf. [Kou11] and [HEK05]).

	Alternatives					
Repeat	1	2	...	j	...	k
1	y_{11}	y_{12}	...	y_{1j}	...	y_{1k}
2	y_{21}	y_{22}	...	y_{2j}	...	y_{2k}
i	y_{i1}	y_{i2}	...	y_{ij}	...	y_{ik}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\ddots$	$\vdots$
n	y_{n1}	y_{n2}	...	y_{nj}	...	y_{nk}
Column means	$\overline{y}_{.1}$	$\overline{y}_{.2}$	...	$\overline{y}_{.j}$	...	$\overline{y}_{.k}$

Table 4.1.: Sample table for the organization of the input to the ANOVA analysis (source: [Kou11]).

The sample table contains k alternatives as columns. For each of these k alternatives n repeated measurements as rows are shown. This leads to a table containing $n \cdot k$ cells. In a first step the column means are calculated for each alternative:

$$\overline{y}_{.j} = \frac{1}{n} \cdot \sum_{i=1}^{n} y_{ij}$$

Using this column mean, each of the measurements in the table can be expressed as the column mean plus an individual error e_{ij}:

$$y_{ij} = \overline{y}_{.j} + e_{ij}$$

In a second step, the overall mean $\overline{y}_{..}$ is calculated by averaging all values:

$$\overline{y}_{..} = \frac{1}{n \cdot k} \cdot \sum_{j=1}^{k} \sum_{i=1}^{n} y_{ij}$$

Using this overall mean, the column means can be expressed as the overall mean plus an effect α of this alternative:

$$\overline{y}_{.j} = \overline{y}_{..} + \alpha_j$$

This leads to the following representation of the single measurements:

$$y_{ij} = \overline{y}_{..} + \alpha_j + e_{ij}$$

This representation shows the idea of ANOVA: Each individual measurements is influenced by the effect of the alternative and an error. In the following steps, these two influencing factors are compared by calculating three so–called sums of square of differences:

$$\text{SSE} = \sum_{j=1}^{k}\sum_{i=1}^{n}(e_{ij})^2$$
$$\text{SSA} = \sum_{j=1}^{k}(\alpha_j)^2$$
$$\text{SST} = \sum_{j=1}^{k}\sum_{i=1}^{n}(y_{ij} - \overline{y}_{..})$$

In these definitions, SSE characterizes the variation which is caused by the errors. In contrast to that SSA characterizes the variation which is caused by the effects. SST can be seen as the total variation. It can be proven (cf. [Kou11]) that $SST = SSA + SSE$. If the differences between the measurements are caused by real differences between the alternatives, SSA should be statistically significant bigger than SSE. To check this, the mean square sums are calculated by dividing the sums of squares defined above by the respective degrees of freedom:

$$\text{MSSE} = \frac{SSE}{k \cdot (n-1)}$$
$$\text{MSSA} = \frac{SSA}{k-1}$$
$$\text{MSST} = \frac{SST}{k \cdot n - 1}$$

An F–test is used to compare the ratio of variances:

$$F_{\text{calculated}} = \frac{\text{MSSA}}{\text{MSSE}}$$

This calculated value is compared to the expected value $F_{\text{table}} = F_{[1-\alpha;(k-1);(k\cdot(n-1))]}$. This expected value can be retrieved from a precalculated table for every confidence level α.

If now $F_{\text{calculated}} > F_{\text{table}}$ then one can conclude that with a confidence of $(1-\alpha)\cdot 100\%$ the variation due to the actual differences between the alternatives is statistically bigger than the variation due to the variation caused by errors. In this way ANOVA can be used to check if there is a statistically significant difference between the alternatives.

This ANOVA pattern can be extended to two or more factors as shown by Kounev [Kou11]. Factors in this context represent group of alternatives and can be regarded as the parameters. Even adding interactions between the factors can be done using the scheme specified above.

ANOVA can be used in two ways: On the one hand it can be used to calculate how likely a parameter has an influence on the variation of the response time. This is no quantification of the influence. So a high probability of influence does not mean that the parameter has a high influence on the variation. It is only very likely that it has some influence, maybe even a very small one. On the other hand, Kounev [Kou11] suggests to use the sum of squares of each parameter to judge on the quantitative influence. A relative quantification can be made by dividing the sum of squares of an individual parameter by the total sum of squares.

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
a	3	3.37	77.55	1.12	460.26	0.0000
b	2	0.52	11.85	0.26	105.50	0.0000
a:b	6	0.43	9.93	0.07	29.46	0.0000
Residuals	12	0.03	0.67	0.00		

Table 4.2.: Example ANOVA result for an analysis of two parameters with interactions (based on [Kou11]).

Table 4.2 contains the result of an ANOVA analysis on an example data set containing two parameters a and b and a response variable. The last column, labeled as "PR($> F$)", contains the probability that the parameter has no effect on the variation of the result. As this probability is very low, one can assume that the two parameters and their interaction have an influence on the result. By calculating the ratio of the sum of squares of each parameter of the total sum of squares the influence on the variation can be quantified: The total sum of squares is $3.37+0.52+0.43+0.03 = 4.35$. The influence of parameter a on the variation is $3.37/4.35 \cdot 100\% \approx 78\%$, the influence of parameter b is $0.52/4.35 \cdot 100\% \approx 12\%$ and the effect of the interaction term of both parameters is $0.43/4.35 \cdot 100\% \approx 10\%$. Some part of the variation can not be explained by the parameters: $0.03/4.35 \cdot 100\% \approx 1\%$. As all parameters and their interactions have been included in the ANOVA, it can be concluded that this part of the variation can be explained by measurement errors. All these values can also be found in the *rSum Sq* column. It should be noted that all these values only apply to this specific model. If parameter b had been removed, the ANOVA would return different results for the influence of parameter a and the residuals.

4.3. Regression Techniques

Regression analysis is a field of statistics which focuses on the relations between some independent variables and some dependent variables. This field includes many regression techniques. These techniques can be used to predict values by using statistical models which were generated before. Many regression techniques exist and this field of study is constantly evolving. New methods are developed and tested for various uses. For this thesis four regression techniques are chosen for the later storage performance modeling and prediction. In the next sections, each of these techniques is discussed in detail. Additionally, each section contains references for in–depth study of the technique. The first three techniques are chosen because of their widespread use by other researchers and their relative simplicity. The fourth technique, M5, was selected because it has proven to lead to exceptionally good results.

4.3.1. Linear Regression

Linear regression [HTF11, p. 41] is a very simple but yet powerful prediction technique. It is the oldest and still the most important tool for modeling. For a given input vector $\vec{x} = (x_1\ x_2\ x_3\ \dots\ x_n)$ and an output variable y a model formed by a linear function is created:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

In this model, β_0 is called the intercept and $\beta_1, ...\beta_n$ are called the coefficients.

If a new x_0 is inserted at the beginning of $\vec{x}$ with a constant value of 1, the coefficients $\beta_0, \beta_1, \beta_2, ..., \beta_n$ can be joined as a vector $\vec{\beta}$ and the whole model can be written as a vector multiplication where $\vec{x}^\intercal$ denotes the transposed vector:

$$y = \vec{x}^\intercal \cdot \vec{\beta}$$

The question arises how to fit this linear model to a set of training data and therefore how to pick the coefficients β. To find the coefficients β and fit the linear model a set of input vectors $X = \{\vec{x_1}, \vec{x_2}, \ldots, \vec{x_m}\}$ and a set of output values $Y = \{y_1, y_2, \ldots, y_m\}$ of equal length is used. Each of the members of the input vector set X is a vector which is defined in the same way as $\vec{x}$ above.

There exist many different techniques for fitting the data, the most popular is the *method of least squares*. This methods picks the coefficients β in such a way that the residual sum of squares (RSS) is minimized. The RSS is defined as:

$$RSS(\beta) = \sum_{i=1}^{m} (y_i - \vec{x_i}^\intercal \beta)^2$$

As RSS is a quadratic function its minimum always exists although it may not be unique. To find this minimum RSS can be transformed into the following matrix multiplication. For this multiplication $\mathbf{X}$ is a $m \times n$ matrix where each row contains one of the input vectors from the set X defined above. In the same way $\vec{y}$ is a vector with m rows containing the output variables from the set Y.

$$RSS(\beta) = (\vec{y} - \mathbf{X}\beta)^\intercal (\vec{y} - \mathbf{X}\beta)$$

The derivative of RSS must be zero for β, leading to the minimum RSS.

$$\mathbf{X}^\intercal (\vec{y} - \mathbf{X}\beta) = 0$$

To calculate β, this equation can be transformed as follows if $\mathbf{X}^\intercal\mathbf{X}$ is non–singular:

$$\beta = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \vec{y}$$

This equation can be finally used to calculate the coefficients for a training set. These coefficients minimize the residual sum of squares. An example linear model fitted using the method of least squares can be found in Figure 4.3. This figure also contains the samples which were used to fit the model. The coefficient β_1 of the input variable x_1 is 1.293 and the intercept β_0 is -1.844. Therefore the fitted linear model can be described as $y = -1.884 + 1.293x_1$.

The linear regression can be extended in multiple ways. Two often used extensions are explained here:

- *Interactions*: Interactions occur in regression analysis if two or more variables do not have an additive effect on the output variable. Instead the effect of one of the variables depends on the value of the other, they therefore cannot be regarded independently.

  To model the interactions between two input variables x_1 and x_2, a new input variable $x_{1,2} = x_1 \cdot x_2$ is introduced. The value of this interaction variable can be calculated by multiplication of the two input variables. The linear regression techniques described above can be used on this input data and compute a coefficient for the interaction term. For the example mentioned in the introduction to this section, including interactions leads to the following linear model:

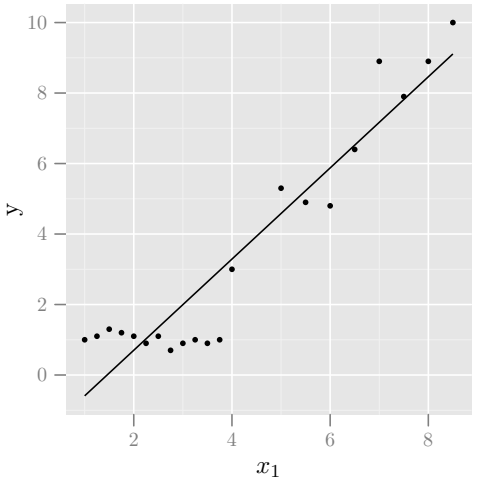


Figure 4.3.: Example linear regression (straight line) of a sample set containing one input variable x and output variable y (printed as dots).

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 (x_1 \cdot x_2)$$

Interactions between more than two variables can be modeled in the same way by simply multiplying all variables which should be included in the interaction. Typically, if interactions are needed for a linear model, all interactions between the variables are included in the model and not only a subset. This means that for a linear model with three variables x_1, x_2, x_3, including interactions between two variables means that three new variables get introduced: $x_{1,2}, x_{1,3}, x_{2,3}$. If also including interactions between three variables, another variable $x_{1,2,3} = x_1 \cdot x_2 \cdot x_3$ has to be added to the linear model.

- *Transformation*: The input variables can be transformed using mathematical functions. The technique described above does not have to be adapted to this transformations. For example if an input variable x_1 is squared before the regression analysis, still a coefficient is found. In this example the linear model is:

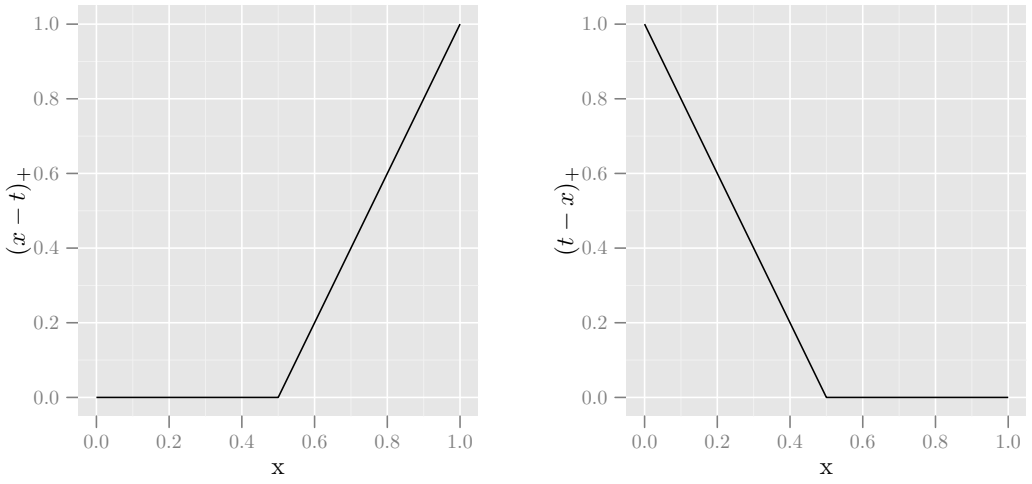
$$y = \beta_0 + \beta_1 (x_1)^2$$

4.3.2. MARS

Multivariate Adaptive Regressions Splines (short MARS) (cf. [HTF11, p. 283] and [Fri91]) is a regression technique which can be seen as an extension to linear models. It allows piecewise linear models. These models can provide a good fit for high dimensional data and are still simple to generate, interpret and predict. MARS is therefore a good choice for modeling performance results [WHKF12] and storage performance results in particular.

MARS builds its model from a collection of basis functions. These basis functions can have two forms:

$$(x-t)_+ = \left\{ \begin{array}{ll} x-t & \text{if } x > t \\ 0 & \text{otherwise} \end{array} \right\}$$
$$(t-x)_+ = \left\{ \begin{array}{ll} t-x & \text{if } x < t \\ 0 & \text{otherwise} \end{array} \right\}$$



(a) First basis function $(x-t)_+$ (b) Second basis function $(t-x)_+$

Figure 4.4.: MARS basis function pair for $t = 0.5$.

In this basis functions, x is one input dimensions and the constant t is the so–called knot. These basis functions are also called hinges due to their look. The two basis functions for $t = 0.5$ can be found as an example in Figure 4.4.

The basis function collection $\mathcal{C}$ contains a pair of basis functions for each input variable X_j with knots at each observed value x_{ij}.

MARS generates a regression model in two steps: The first step is the creation of a large intermediate model. This is also called the forward step. In the second step, also called the backward step, the model is pruned to a smaller model.

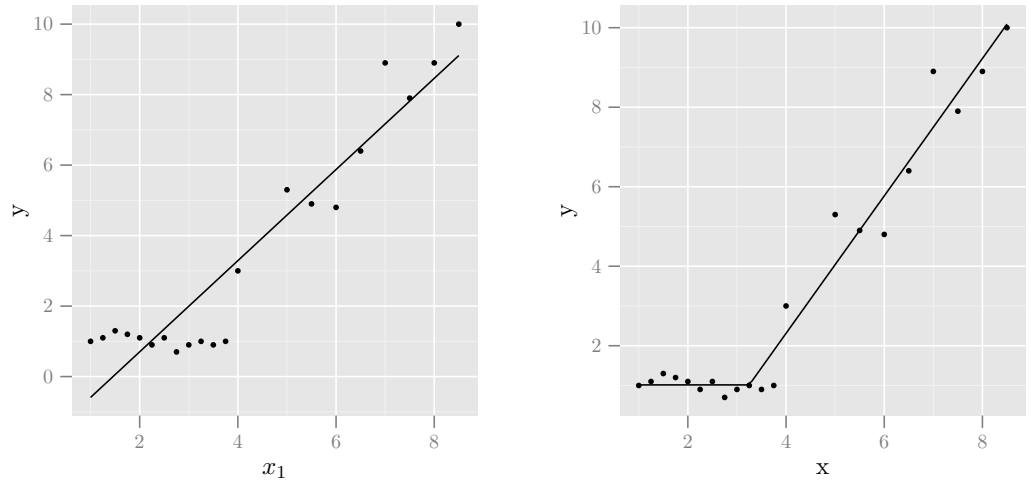
For the first step MARS iteratively adds a new pair of basis functions to the model. These basis functions are simply added to the model from the previous iteration. The basis functions are chosen from basis function collection $\mathcal{C}$. The algorithm may choose to multiply the selected basis function with basis functions already existing in the model. This additional multiplicator allows higher dimensional data to be modeled. The algorithm selects the basis function and the eventual multiplicator which reduce the training error the most. This procedure is repeated until either the maximum of terms allowed in the model is reached or the training error gets below a certain threshold.

The output of the first step is typically overfitted and so the backward step deletes terms from the model. In contrast to the first step, where always pairs of basis functions have been added to the model, the backward steps can remove single basis functions. The backward steps iteratively deletes those basis function whose deletion causes the smallest increase in the training error. This training error is weighted using the model complexity. This means that the term which has a high complexity is removed first if this removal does not reduce the training error to much. This process is repeated until the model contains less model terms than a configured limit.

Because of the linearity of the basis functions a MARS model can be computed efficiently. Also the prediction of unknown values can be done very fast due to its easy computation.

MARS contains some configuration parameters which must be set by the user prior to its use. The influence of these parameters on the storage performance prediction models is evaluated in a later chapter.

- *Maximum degree of interaction*: This number specifies how many terms may be included in the multiplications in the forward steps. By setting this configuration to



(a) Linear Model (b) MARS Model

Figure 4.5.: Comparison between a linear regression model and a MARS model fitted to the same data. The regressions are shown as straight lines, the sample set is printed as dots.

1 no multiplication is allowed and therefor no interaction can be modeled. On the one hand, this reduces the quality of the model, on the other hand, the complexity of the model is also reduced.

- *Maximum number of terms in the forward step*: This limits how many terms may at most be included in the model after the forward step. The larger the model gets, the more computation is involved in the first step. This also increases the computation time when removing terms in the second step. On the other hand a larger input model to the backward step allows the backward step to select from more terms and therefore calculate a better model.
- *Threshold for the forward step*: As explained above, the forward step can come to an end if the training error falls below a certain threshold. This threshold can be configured. The same argumentation as before holds for the amount of terms in the model after the forward step.
- *Number of terms in the pruned model*: This specifies how long the backward step should be repeated: The step is stopped if there are not more terms in the model than this threshold. This settings allows the configuration of the final model complexity and must be chosen wisely: A too big value makes overfitting more likely, a too small number makes a worse prediction model. It makes no sense to set this value higher than the maximum number of terms in the forward step.

Figure 4.5 contains the comparison of a linear regression model and a MARS model fitted to the same data. It can be seen that the MARS model can model the hinge character of the data due to its basis functions whereas the linear model can only use a single coefficient.

4.3.3. CART

Classification and regression trees (short CART) (c.f. [BFSO84] and [HTF11, p. 267]) are a group of algorithms which all have their usage of trees to model the data in common.

Figure 4.6 contains an example regression tree for two input variables. To predict a new, unknown value, the evaluation starts at the root and the condition in this node is checked.

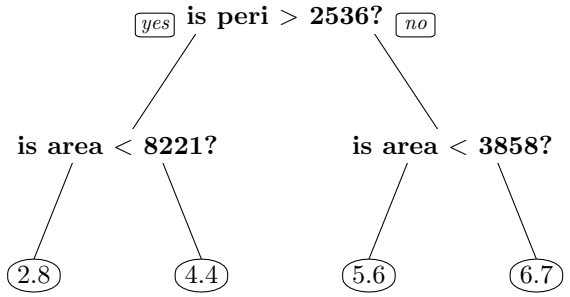


Figure 4.6.: Example for a regression tree containing two input variables ("area" and "peri") and an output variable.

If the condition is true, the left edge is followed, otherwise the right edge. This is repeated until a leaf is reached. The leaf contains the value to predict. Regression trees are in most cases binary trees with conditions in their non–leaf nodes and values in their leaf nodes. [HTF11, p. 273] states that non–binary splitting is not a good generic choice: Although it might help for some special problems, in general the data gets fragmented to fast if using multi way splits.

The biggest advantage of CART models is the easy interpretability even for high dimensional data and their fast prediction due to their simple nature. A disadvantage of regression trees is their inability to use combinations of variables or to model interactions. Also it is not possible to include additive effects in the models. The model might also be misleading: A parameter might either not be included in the model creation or might be left out after the model creation. Both times it does not show up in the regression tree so no distinction is possible.

The main challenge is the creation of a regression tree which includes the fitting of a regression tree model to a training sample set. Similar to MARS explained above, the algorithm explained by Hastie et al. [HTF11] and used in this thesis is split into two steps: The first step creates a potentially overfitted tree which is pruned to a reasonable size in the second step. The two steps are explained in detail in the next paragraphs.

If the tree has n leaves, $L_1, L_2, \ldots, L_n$, the output or the prediction for the whole tree is:

$$f(x) = \sum_{i=1}^{n} (avg(L_i) \cdot I(x \in L_i))$$

Where $avg(L_i)$ denotes the average of the output variables of all samples which are represented by this leaf. $I(x \in L_i)$ is 1 only if $x \in L_i$ which means that x is represented by the leaf L_i.

This leads to the first step of the regression tree fitting, the creation of the tree: An initial tree which only contains a single node is created. This node represents the whole training data and therefore predicts the mean of the whole training data output. The algorithm runs in an iteratively way: In each step, all leaves in the tree are split into two new leaves. To do this split, the algorithm has to find a splitting variable j and a splitting point s. If these two parts have been found, the old leaf L_i is split into two new leaves L_{i1} and L_{i2} where L_{i1} represents all data from L_i where $j \leq s$ and L_{i2} represents all data from L_i where $j > s$. The searching of j and s is done by looking at each of the input variables and calculating the best split point s for this variable. The split point is set in such a way

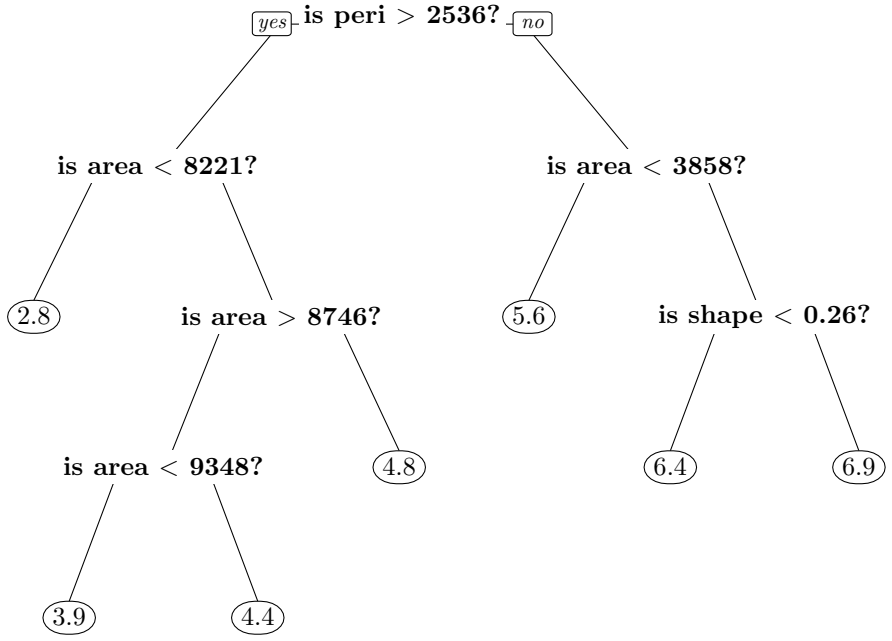


Figure 4.7.: Regression tree before pruning: This figure shows the output of the first step which will be pruned in the second step of the regression tree generation. Compare to Figure 4.6 which shows the same tree after the pruning step.

that the mean squared error is minimal. For each of the input variables the mean squared error of their best split is compared and the lowest value is selected. This leads to the input variable j and the splitting point s which produce the smallest error.

This splitting is repeated for each of the leaves in the tree. The algorithm stops to split a leaf if it contains less samples than a preconfigured threshold (typically 5) or if the split does not improve the error of the whole tree by at least a configured value (typically 0.01). For the whole tree the algorithm is stopped if no leaf can be split any further.

As mentioned above, this splitting leads to a potentially overfitted tree. For this reason in a second second step the tree is pruned by merging or collapsing non–leaf nodes. This is done by calculating a cost criterion for the tree: This criterion contains the quality of the model and the amount of nodes in the tree. By adding a weight term α to this criterion the trade off between the two parts can be configured. For a given value for α, the tree which minimizes the cost criterion can be found using weakest link pruning [HTF11]. This minimal tree is unique and always exists. To select an appropriate value for the tuning parameter α, a general cross–validation (see Section 4.5 for a detailed explanation) is used: The training data is split into five or ten groups. For each of the groups and each of the potential values of α the pruning explained above is executed. The α which minimized the sum of squares is chosen and the final tree is computed.

Figure 4.7 shows the same tree as Figure 4.6 but before the pruning step. You can see that the pruning step reduced the complexity of the tree from 13 to 7 nodes.

The CART algorithm has two major configuration variables which play a role for the parameter analysis later in this thesis. Both parameters modify the behavior of the first step. The pruning step has no external configuration because it finds its optimal setting for the α variable using cross–validation.

- *The minimum numbers of observations to try a split*: The smaller this number is, the more nodes are generated in the first step of the algorithm. Decreasing this parameter therefore increases the tree size before the pruning step. Typical values range from 5 to 20 depending on the total number of observations.
- *The complexity parameter*: Any split which does not increase the error by at least the value of this parameter is not done. This parameter can be set to 0 which disables this check and only checks for the minimum number of observations for the split. Setting larger complexity parameters reduces the work which needs to be done in the pruning step because the output of the first step is smaller.

4.3.4. M5

M5 [Qui92] is a regression technique which combines the benefits of linear models with those of CART. The idea is to add a linear model to each leaf of a regression tree. The final M5 model consists of a decision tree, where each of the non–leaf nodes contains a condition (also called a test) and the two edges starting from this non–leaf node reference another node which should be evaluated if the condition is true for the one edge and false for the other edge. The leaf nodes of the tree contain a linear regression model.

The algorithm to fit such an M5 model to the training data consists of two steps. This is again similar to MARS and CART: The first step generates a potentially overfitted tree and the second step removes the nodes which are not necessary for a good fit.

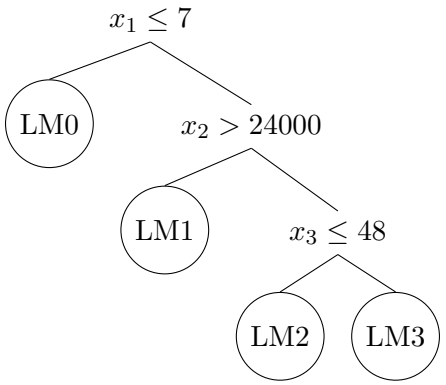
The first step takes an iterative approach similar to the first step of CART: It starts with a tree consisting of a single node representing all training data. Now the following steps are repeated for each leaf in the tree:

All samples which are represented by the leaf under consideration are named T. The first step is to find a condition which splits the samples into two new sample sets T_0 and T_1, where T_0 contains all those samples which do not meet the condition and T_1 contains those sample which meet the condition. To find the condition, each possible condition is evaluated: For each condition, the two sets T_0 and T_1 are computed. The standard deviation is then calculated for each of the sets. The algorithm selects the split which maximizes the reduction of the standard deviation.

This split is saved and two new leaves are appended to the original node. These two leaves represent the samples T_0 and T_1. For each of the two leaves, a linear regression model is fitted to all samples in T_0 or T_1. To reduce the complexity of the linear models which are attached to the leaf, the model is simplified. This is done by removing parameters from the linear model. To find these parameters, the estimated relative error is defined as $((n + v)/(n - v)) \cdot \text{relativeError}$. Here n is the number of samples in T_0 or T_1 and v is the number of parameters in the linear model. This definition extends the relative error with a complexity term. The parameters are removed from the linear model until this estimated relative error is minimized. The definition of the estimated relative error leads to the fact that a term can be removed although it increases the (ordinary) relative error as long as the complexity is decreased by an adequate value.

After this simplification, the splitting of the nodes is iteratively repeated until one of the following conditions is met: A node is not split further if it represents less samples than a predefined limit. Additionally, no split is made if the standard deviation of the samples represented by the node is less than a configured threshold.

The first step leads to a tree which has a linear model attached to every node. The final model will only contain a linear model in the leaves. This transformation is done in the second step together with the pruning of the tree.



(a) Regression tree

Model	LM0	LM1	LM2	LM3
(Intercept)	11.5	-10.1	11.9	217.5
x_1			0.03	
x_2	0.902			
x_3				
x_4	0.518	34.6		

(b) Linear models coefficients

Figure 4.8.: Example M5 model consisting of the regression tree and the attached linear models. The left edge of a node in the regression tree must be evaluated if the condition is true, the right edge if the condition is false.

The pruning of the tree examines each of the non–leaf nodes in the tree, starting at the bottom. Again using the definition of the estimated relative error, two scenarios are evaluated for each node:

The first scenario means keeping the sub tree of the node: This has an estimated relative error of $((n + v)/(n - v)) \cdot \text{relativeError}$, where n is the number of samples represented by the whole sub tree and v is the number of parameters in the sub tree, both in the conditions and the linear models. The relative error is the relative error of all samples in the sub tree when using the their appropriate linear models.

The second scenario, the removal of the sub tree, making the examined node a leaf itself. The estimated relative error is again calculated as $((n + v)/(n - v)) \cdot \text{relativeError}$. This time, n is the number of samples represented by the newly generated leaf. This number is the same as in the other scenario. v is the number of parameter used in the leaf, either by its condition or by its linear regression model. The relative error is calculated based on the linear model which is assigned to the current node. As explained above, each of the nodes has a liner model attached to it.

For each node, the estimated relative errors for both scenarios are calculated and the scenario which leads to a smaller error is chosen. This is repeated for each node in the tree. In a last step, the linear models of all non–leaf nodes are removed. These models are not needed anymore.

The two aborting conditions for the forward step are defined and calculated by the algorithm and cannot be set by the user of M5. This means that the M5 algorithm has no configuration options.

Figure 4.8 shows an example for an M5 model. The model uses four variables $x_1, \ldots, x_4$. Not all variables are included in the regression tree: x_4 is only used in the linear models. The LM3 linear model is only a constant term. This shows that all variables have neither to be included in the regression tree nor in the linear models. Some variables can even be purged from both parts and not be used at all.

4.4. Model Comparison Metrics

In the following section, different metrics are discussed. These metrics can be used to judge on the quality of a statistical model and to compare different models and regression techniques. In this context the following sets, which are of equal length, are defined:

$$A = \{a_1, a_2, ..., a_n\}$$
$$P = \{p_1, p_2, ..., p_n\}$$

The A set contains the actual dependent values or the measurement results. The P set contains the matching values which were predicted by a statistical model. This means that by using different regression techniques, only the content of the P set changes.

4.4.1. RMSE

The *Root Mean Square Error* metric is calculated using the following term:

$$\text{RMSE}(A, P) = \sqrt{\frac{\sum_{i=1}^{n}(a_i - p_i)^2}{n}}$$

The unit of the RMSE is the same unit as the actual measurements. The metric penalizes big errors and provides an estimate for the size of a "typical" error.

4.4.2. MAE

The *Mean Absolute Error* also uses the same metrics as the measurements. It is always smaller or equal than the RMSE because of the missing square term. For the same reason, it does not penalize big errors. The MAE is calculated using the following formula:

$$\text{MAE}(A, P) = \frac{\sum_{i=1}^{n}|a_i - p_i|}{n}$$

4.4.3. MAPE

The *Mean Absolute Percentage Error* expresses the quality of the model using a percentage value. This makes it easier to understand because value ranges do not play a role here. The following term is used to calculate the MAPE:

$$\text{MAPE}(A, P) = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{a_i - p_i}{a_i}\right|$$

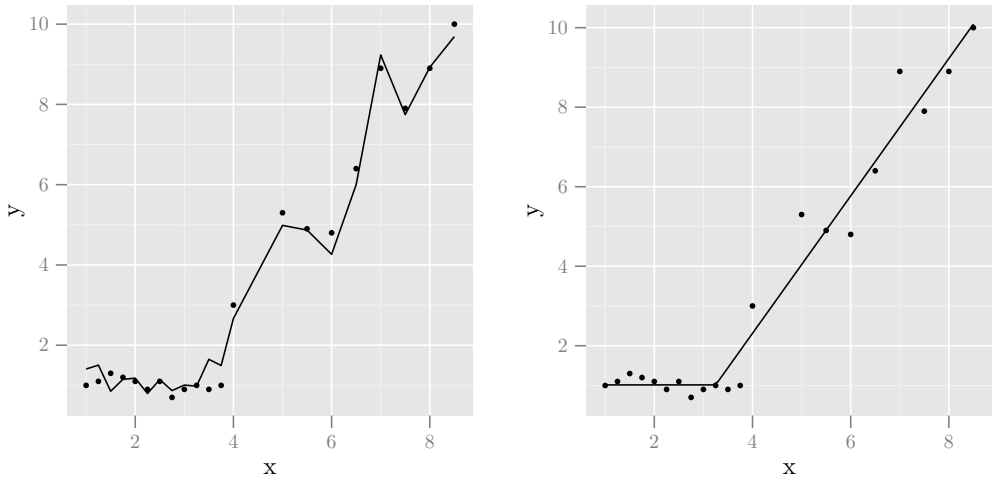
4.4.4. Coefficient of Determination

The *Coefficient of Determination* is often called R^2. It is widely used by many different authors because it allows an easy interpretation how good the model fits the actual values. Kvalseth [Kva85] shows that there are many different definitions of this coefficients. These definitions result in different values for the coefficient. This makes comparing different publications from different authors difficult without the knowledge about the actual R^2 definition used. Additionally, the author explains that R^2 is not suitable for comparing the results between different regression techniques. For this reason the coefficient of determination is not used during this thesis.

When used, the following definition is typically applied:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(a_i - p_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$$

This means that the R^2 can be used to judge the goodness of the fit in comparison to the mean. While for very poor predictions the R^2 can become negative it never becomes bigger than 1 due to the squares in the fraction.



(a) Overfitted MARS model (b) Not overfitted MARS model

Figure 4.9.: Example for an overfitted model on the left and a non overfitted model using the same samples on the right.

4.5. Cross–Validation

Cross–validation (c.f. [HTF11, p. 214] and [Ste07]) is a simple and widely used solution to estimate the prediction error. It can be used as a solution for three different problems:

- The judgment on the quality of a single regression model.
- The comparison of the quality of different regression techniques. The models must be built from the same data set to make a comparison possible.
- The comparison of the quality of different configurations of a single regression technique. This can be used for example to check if increasing the degree of interaction improves a MARS model.

Given a set of available samples $\mathcal{S}$ which should be used to answer one of the questions above, a trivial solution would be to use the whole sample set $\mathcal{S}$ to train the models and then check for each sample in $\mathcal{S}$ again how good the model predicts the original output. This method falls short of multiple points: Typically, the quality of the model can be defined as the generalization ability. Generalization means that the model adapts to the underlying structure of the data and not to the data itself. By testing with the same data which was used for training, the generalization ability cannot be tested. This leads to the second problem of this approach: The model can be heavily overfitted. An overfitted model describes only the random effects in the measurements and not the underlying data. Figure 4.9 contains an overfitted regression model: Although it has a very small error for the training data, it does only model the random effects of the data and not the underlying structure. The right model shows a better fit for the samples. Although the error is larger when being tested using the same samples as used for training, the model must be considered to be better in terms of generalization ability.

To test one or multiple models for their generalization abilities or compare the generalization abilities of multiple models, cross–validation can be used. This approach is based upon the separation of the samples into two groups: The training and the test set. These two sets are disjoint sets, so no sample is in both, the training and the test set. The separation is done before the model generation. Then the regression model is trained using the training data. Afterward, the model is tested using the test data and an appropriate metric (see Section 4.4).

There are multiple ways to split the data into the training and the test set. One of the most used approaches is the so–called k–fold cross–validation. Figure 4.10 shows the steps which are taken for the k–fold cross–validation. These steps are discussed in detail in the following paragraphs.

Given a sample set $\mathcal{S}$ of length n, k can be in the range $1 < k \leq n$. Setting $k = 1$ would lead to no cross–validation at all. In a first step the sample set is randomized meaning the samples are brought in a random order. The second step cuts the sample set into k folds of roughly equal length. This is done to prepare the cross–validation. For each model or each regression technique which should be compared or judged the following procedure is executed for each $i \in \{1, ..., k\}$:

The model is trained using all folds except the i^{th} fold. This means that the model is fitted to $k - 1$ folds. The i^{th} fold, which was left out in the training is used in the next step to test the model. This means that the model comparison metrics (see 4.4) are applied to all samples in the i^{th} fold.

Due to this approach the cross–validation returns k different values for each model metric, one for each fold. This process ensures, that new, unused samples which have not been used in the training process are used to validate the model. Therefore cross–validation and k–fold cross–validation in particular effectively prevent overfitting.

The case $k = n$ is known as leave–one–out cross–validation: For each sample in the data set a new model is built without this sample and the single sample is used to calculate the model comparison metrics. The computational overhead of this calculation must not be underestimated as n models need to be created.

The question how to choose k cannot be answered in general [HTF11]. Large values up to $k = n$ lead to a high variance because the training sets are very similar. As explained before, the computation time is also high. Smaller values for k, for example $k = 5$ have a lower variance. Especially for small data sets, there is a bias in cross–validation with small k. This may lead to an overestimation for the true error depending on the regression technique.

Independent of the value chosen for k, cross–validation can only provide an estimate for the true error for a model trained using the whole data set $\mathcal{S}$. K–fold cross–validation solves the problem of cross–validation even if the data set is very spare because it reuses every sample for training and testing.

There are other methods for preventing overfitting, for example bootstrapping [HTF11, p. 217]. The authors of this book show that bootstrapping roughly performs as good as cross–validation in terms of error prediction and stability.

During this thesis cross–validation is used extensively to provide accurate estimations on the quality of models and regression techniques.

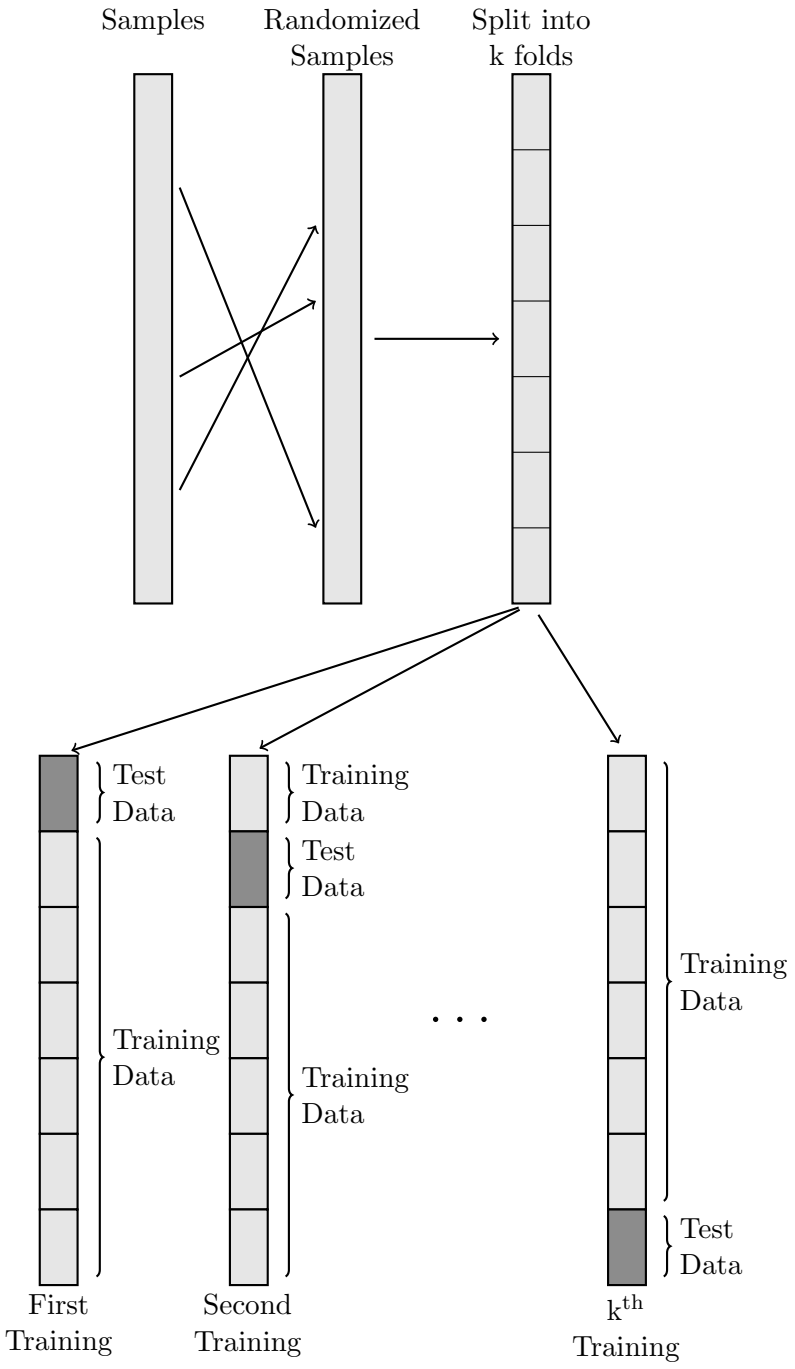


Figure 4.10.: Steps necessary for k–fold cross–validation.

5. Experimental Methodology

This chapter explains the methodology which is used to obtain the benchmark results. The actual results are discussed in the later chapters. This chapter starts with a discussion on the assumptions which are made and then continues with the tools which are used to obtain the results.

5.1. Setup

The whole benchmarking is done on the IBM System z and the IBM DS8700 as they are described in Chapter 3. The important information, which had to be decided on, were the parameters which should be varied and the values for each of these parameters. This approach resembles the one presented by Noorshams et al. [NKR12], who also use the same system and the same benchmarks.

As the parameter space is very huge, some general assumptions on the setting of parameters have to be made: As mentioned in Section 3.3, some parameters have to be regarded fixed due to inabilities to modify the hardware. This includes the RAID settings and other internal settings of the IBM DS8400 as changing them cannot be done without data loss which was unacceptable in the scenario of this thesis.

Other parameters are set to fixed values to reduce the parameter space and therefore reduce the benchmarking time for the thesis. Nevertheless these parameters could be changed and evaluated using the same approach specified here. Even the tools, which were written for this thesis and which are discussed later, support changing of most of the parameters which are theoretically possible. Nevertheless, varying more parameters automatically increases the amount of experiment which have to be carried out.

The following parameters are set to fix values for the whole thesis:

- *File size*: The file size is set to 16 MB. The influence of changing the file size is not discussed in the thesis.
- *File system*: The file system is set to the ext4 file system. As it is currently the most used file system, the results should apply to typical systems.
- *Operations per file*: Each read and write operation is divided into 256 sub operations of the same type. Depending whether random or sequential access should be benchmarked these 256 sub operations are done sequentially or randomly in one file. For each of the sub operations the response time is recorded independently.

- *Thread count*: The thread count of the benchmark can be regarded in two ways: As a workload parameter or as system configuration parameter. The latter is true because the thread count can be seen as the load which is currently present on the system. As the benchmarking can anyway not be done without any load on the system, a fixed thread count can be seen as a way to model the usage of the system. The fixed value of 100 Threads is chosen because it does not overload the system but there are enough threads to simulate some load. Small numbers for the thread count lead to very unstable results.
- *Caches*: Caches in the operating system are disabled using the `O_DIRECT` flag. As noted in Section 3.3, this does not mean that the caches of the storage system are disabled. As the IBM DS8700 still is a fully cached system, the effects of caches is still included in the models.
- *Virtualization technology*: Due to time constraints and technical difficulties the virtualization technology remains fixed at PR/SM. Compare to Section 3.3 for other possibilities that are not considered in the thesis.
- *Software Versions*: All benchmarks are carried out on a Debian 6.0.2. The Linux kernel version is upgraded to 3.2.0 due to bugs in the default kernel which was shipped with Debian 6.0.2.

Other parameters are varied within their respective ranges. These are also the parameters whose influence on the response time is discussed in later sections.

- *Block size*: The block size is kept in the range from 4 kB to 32 kB. The lower bound of 4 kB is chosen because of the block size of the ext4 file system is also 4 kB. The upper bound is a synthetic bound which is selected for no technical reasons but to limit the range. The block size is explored in 4 kB steps which leads to eight different values for this parameter. The block sizes of read and write requests are kept at the same value for each individual experiment.
- *Read percentage*: The ratio of read and write requests is controlled by the read percentage. For obvious reasons the range can be 0% to 100%. 0% means a pure write workload and 100% stands for a pure read workload. Five values between the two extreme values are selected for a total of seven values for the read percentage.
- *Sequential/Random access*: The switching between sequential and random access is done for both, read and write operations at the same time. Technically it would be possible to have sequential writes and random reads at the same time.
- *Scheduler*: Two schedulers are benchmarked: The CFQ scheduler which is the default scheduler on most Linux systems and the NOOP scheduler which is the recommended scheduler for complex storage systems with their own scheduling logic like the IBM DS8700.
- *File set size*: As explained above the file set size plays an important role for the behavior of the requests. As the IBM DS8700 has a read cache of 50 GB, huge file sets are needed to test the performance of bare hard disk operations. For this reason the file set is set in the range between 1 GB and 100 GB in five steps.

These two lists define a set of experiments which are executed for this thesis. Table 5.1 contains another view on these experiments. Additional benchmark experiments are needed for various sections. The configuration for these experiments is noted at the appropriate places in the sections.

For each benchmarking run, the individual response times of each request are collected. From these individual results the mean response time can be calculated afterward.

Parameter	Values
Block size	4 kB, 8 kB, 12 kB, 16 kB, 20 kB, 24 kB, 28 kB, 32 kB
Read percentage	0%, 25%, 40%, 50%, 60%, 75%, 100%
Access mode	random, sequential
Scheduler	CFQ, NOOP
File set size	1 GB, 25 GB, 50 GB, 100 GB

Table 5.1.: Values of the parameters which are varied in this thesis. These values lead to a total of 1120 configurations when using full exploration.

To test the results for their stability and repeatability, FFSB was run multiple times. For each configuration of the parameters explained above the following steps are taken:

1. *Prepare the file set*: The file set is created with the correct size.
2. *Warm–up run*: A first run of 60 seconds is executed. The results are not collected but thrown away.
3. *Five benchmarking runs*: Five benchmarking runs, each of 60 seconds duration, are executed one after the other. The results of each of these runs are collected.

For each configuration at least $1 + 5 \cdot 1 = 6$ minutes of benchmarking time plus the time for the file set creation is needed. The basic configuration explained above for the two schedulers leads to a total of $8 \cdot 7 \cdot 2 \cdot 2 \cdot 5 = 1120$ configurations which need to be benchmarked on the system under test. The minimal duration for this benchmark run is $1120 \cdot 6 = 6720$ minutes or 112 hours or about 5 days. As mentioned in Section 3.4, the creation of the file set takes a long time. For this reason the total benchmarking needs much more time than this duration.

As the manual benchmarking of the 1120 configurations defined above is a time consuming and error prone task, this process has to be automated. For this reason the *Storage Benchmark Harness* was written. This tool automates the benchmark execution and result gathering. Because the analysis of the results is also time consuming, this part has also been automated in the so–called *Analysis Library*. The following section discusses this automation in detail.

5.2. Experimental Automation

The application of the benchmarking and the analysis on another system is not only tedious, time consuming and error prone, it also requires in–depth statistical knowledge and statistical evaluation expertise. Thus, this thesis takes an approach which includes a highly automated analysis and evaluation support. For this reason two pieces of software, the *Storage Benchmark Harness* and the *Analysis Library* were developed.

Both parts run on the controller machine (also called the measurement machine): The *Storage Benchmark Harness* consists of multiple components: The benchmark controller controls the benchmarks on the systems under test. It uses benchmark drivers to connect to the preexisting benchmarks and retrieve their results from the systems under test. The results are stored in a SQLite[1] database. Additional benchmark drivers can be added easily to the Storage Benchmark Harness. Nevertheless, the Storage Benchmark Harness should not be regarded as a framework.

The *Analysis Library* adds functionality to an already existing analysis software (R[2]). This functionality includes an interface to the SQLite database for data retrieval. Another

[1] `http://www.sqlite.com`
[2] `http://www.r-project.org/`

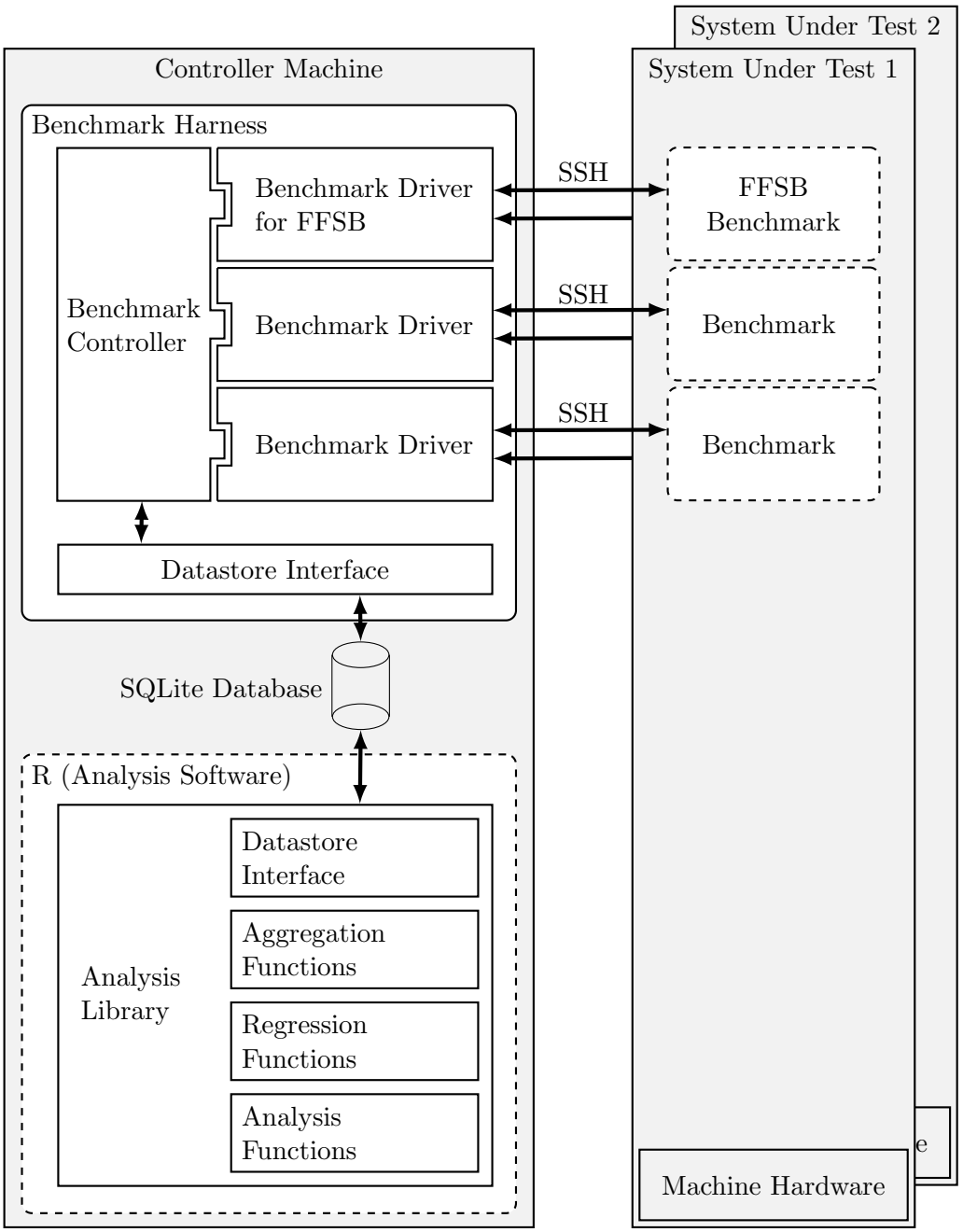


Figure 5.1.: Overview of the components involved. Dashed components are existing and therefore reused. Two additional benchmark have been included in the diagram for demonstration purposes. Currently only an FFSB benchmark driver is available. Other benchmark drivers can nevertheless be added easily.

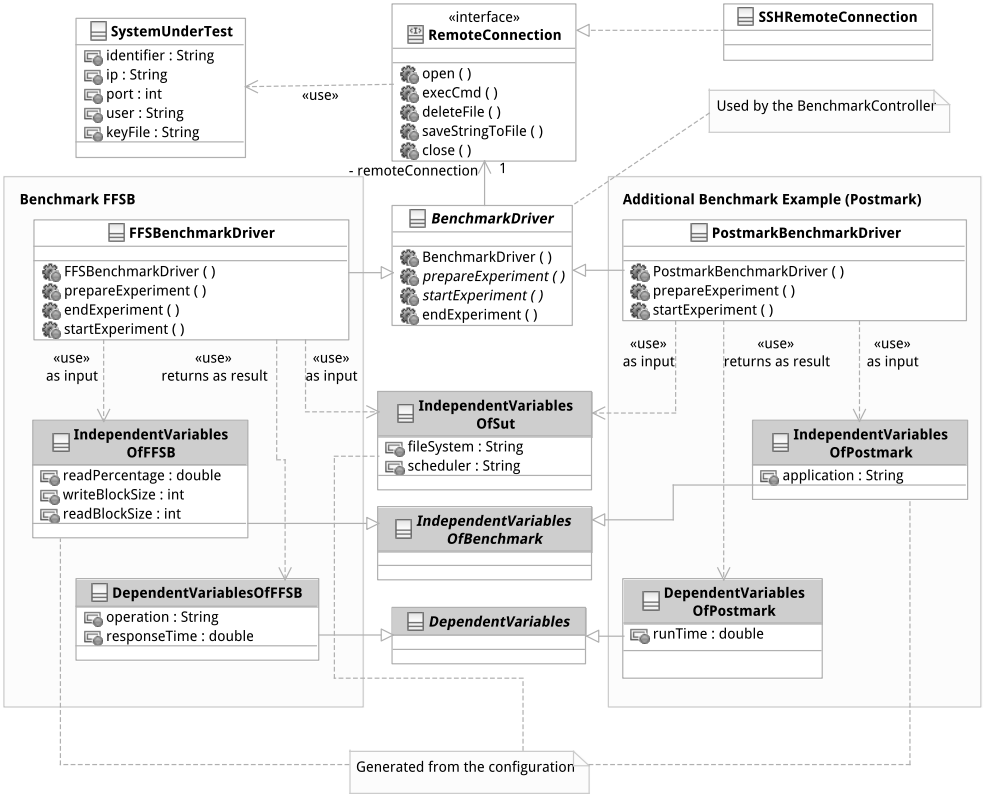


Figure 5.2.: Class diagram for the benchmark drivers and the other classes involved in this context.

important functionality of the Analysis Library is the automated model generation. In this way, the Analysis Library helps to better understand the benchmark results.

Figure 5.1 gives an overview of the components which were developed for this thesis. Currently the Storage Benchmark Harness only includes a single benchmark driver for the FFSB benchmark. The other two benchmark drivers can be seen as place holders. In the following two sections, the two parts of the experimental automation are explained in detail.

5.2.1. Storage Benchmark Harness

The purpose of the *Storage Benchmark Harness* is the automated execution of benchmarks and the gathering of the results. To interface with a preexisting benchmark on the system under test, a so–called `BenchmarkDriver` is used. Figure 5.2 contains an UML class diagram for two example benchmark drivers. As mentioned above only the `FFSBenchmarkDriver` is currently implemented, the second adapter can be regarded as a place holder.

As the Storage Benchmark Harness needs an explicit description what experiments should be run on which systems under test, a configuration is necessary. Figure 5.3 shows the UML class diagram for the configuration of the experiments.

Each benchmark driver takes a set of independent variables as input. These independent variables contain values for each parameter the benchmark driver supports. Additional variables are those parameter which are independent from the benchmark but specific for the system under test, for example the file system. For this reason the `FFSBenchmarkDriver` accepts a set of independent variables for the FFSB (`IndependentVariablesOf-`

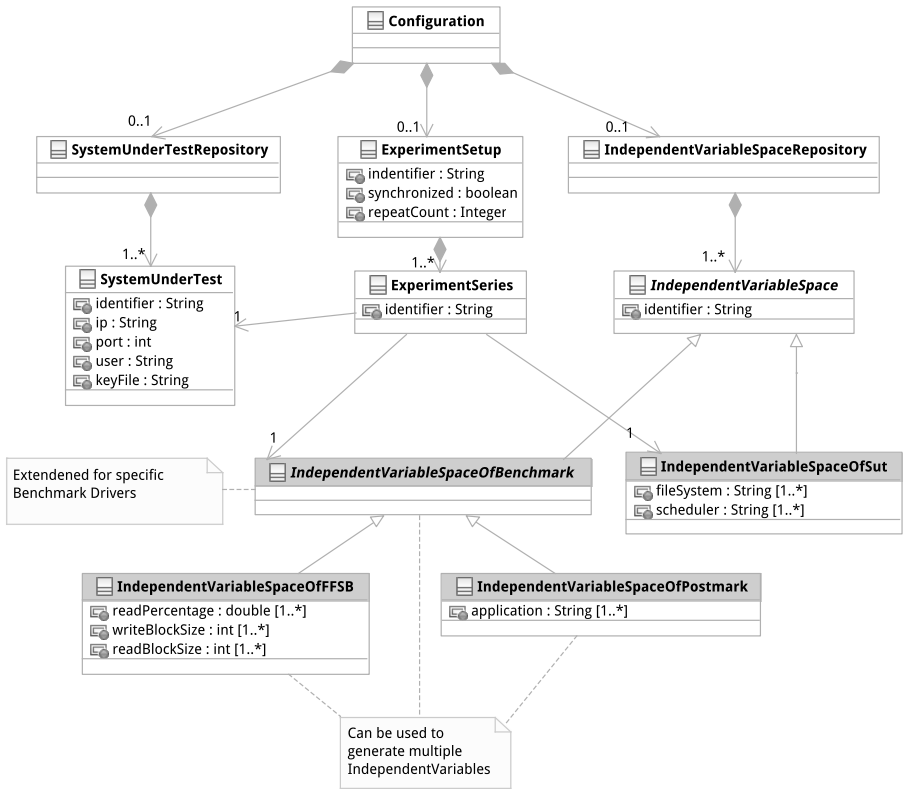


Figure 5.3.: Class diagram for the classes used to represent the configuration for the benchmarking process.

Parameter	Values	
fileSystem	ext4	XFS
scheduler	CFQ	

(a) Independent variable space for system under test

Parameter	Values		
readBlockSize	8 kB	16 kB	32 kB
writeBlockSize	64 kB		
readPercentage	25%	75%	

(b) Independent variable space for benchmark

Table 5.2.: Examples for independent variable spaces.

`FFSB`) and a set of independent variables specific for the system under test (`IndependentVariablesOfSut`).

To define multiple experiments at once, the concept of so–called independent variable spaces is used. An independent variable space contains one or more values for each parameter. Using full exploration, an independent variable space can be used to generate multiple independent variables. Table 5.2 contains two independent variables spaces as example, one for a system under test and one containing values for the parameters of a benchmark.

An experiment series in this context consists of three parts: The independent variable space of a system under test, the independent variable space of a benchmark and the system where all the experiments should be executed. For example, the two independent variable spaces given as example above in Table 5.2 can form an experiment series.

As explained above, the independent variable spaces are expanded using full exploration. This means that all possible combinations of the variable values are produced. As an example, the two independent variable spaces mentioned in Table 5.2 can be expanded

	Independent Variable Values ...				
	... of System Under Test		... of Benchmark		
	`fileSystem`	`scheduler`	`readBlockSize`	`writeBlockSize`	`readPrecentage`
Experiment 1	'ext4'	'CFQ'	8 kB	64 kB	25
Experiment 2	'ext4'	'CFQ'	8 kB	64 kB	75
Experiment 3	'ext4'	'CFQ'	16 kB	64 kB	25
Experiment 4	'ext4'	'CFQ'	16 kB	64 kB	75
Experiment 5	'ext4'	'CFQ'	32 kB	64 kB	25
Experiment 6	'ext4'	'CFQ'	32 kB	64 kB	75

Table 5.3.: Example for the full exploration of an experiment series containing the variable spaces defined above in Table 5.2.

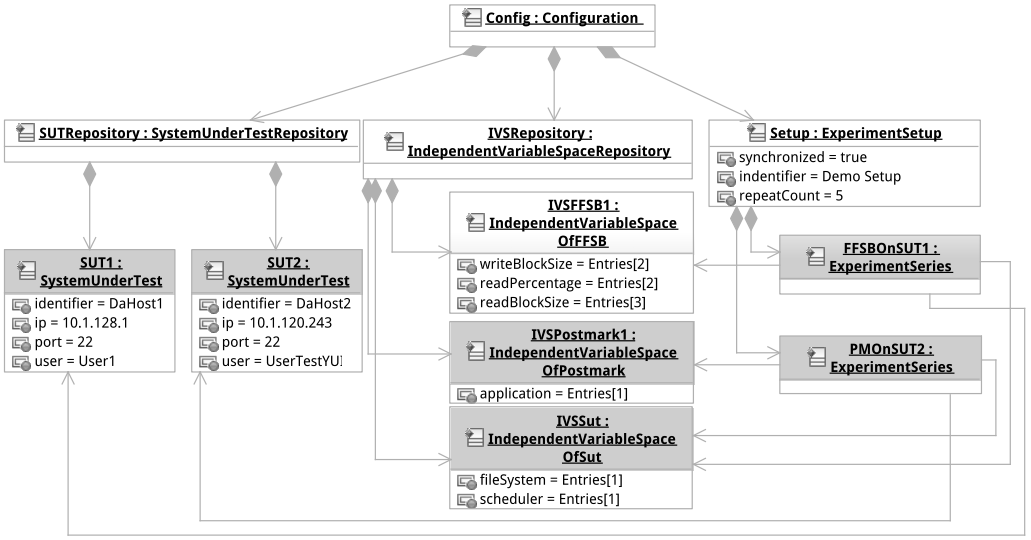


Figure 5.4.: Object diagram of a configuration instance.

to the six experiments listed in Table 5.3.

Figure 5.4 contains an UML object diagram for a sample configuration involving two systems under test which are configured to execute their benchmarks in parallel.

Another important feature of the Storage Benchmark Harness is the synchronized execution of the benchmarks. As explained in the section above, FFSB (and other potential benchmarks) run the benchmark in multiple phases. For this reason the benchmark controller supports three independent phases which are run for each experiment:

- The preparation phase, where the configuration files can be generated on the system under test and the warm–up run can be done. As some benchmarks do not need this functionality in their benchmark drivers, the implementation is optional. The implementation can be done in the `prepareExperiment` method of the benchmark drivers.
- The actual benchmarking, where the benchmark is executed and the results are collected. This happens in the `startExperiment` method of the benchmark drivers.
- A finishing phase, where leftovers can be cleaned up. This is encapsulated in the `endExperiment` method.

The Storage Benchmark Controller of the Storage Benchmark Harness runs each of these methods synchronized on all systems under test. This means that the actual benchmarking happens at exactly the same time on all systems under test. This is important for benchmarking virtualized environments.

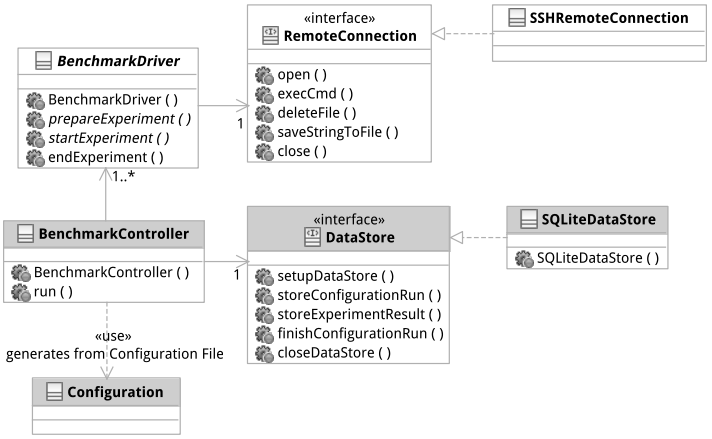


Figure 5.5.: Class diagram for the runnable parts of the harness which includes the controller and the data store.

In a last step, the Storage Benchmark Harness uses a SQLite database to save all the results to the hard disk. The class diagram in Figure 5.5 shows the runtime components of the Storage Benchmark Harness. This also includes the data store interface which is used for the result saving. This enables an analysis of the results independent of the benchmarking runs by the Analysis Library.

5.2.2. Analysis Library

To analyze the results which are gathered by the Storage Benchmark Harness, R is used. As the storage controller saves the results in a SQLite database, the results must be read from there. The Analysis Library additionally contains a set of functions which help to build regression models from the results. Additionally, all the analysis done later in this thesis can automatically be executed. This means that most of the figures and tables in this thesis can be recreated using new data. Additional analysis on the benchmark results or the regression models can be executed using the provided functions and the features of R.

The automated regression model generation uses the caret library [Kuh08]. Using this library the following steps are done in an automated way:

1. The folding which is needed for the cross–validation (see Section 4.5) is done by splitting the input data, for which the regression models should be built, into 10 folds.

2. For each of the regression techniques which should be evaluated, a cross–validation is executed. It uses the folds generate in the step before. Multiple comparison metrics for each regression technique and each model are stored and recorded. This process ensures that all regression models are trained and tested using the same data sets.

3. A final model using all data is generated for each regression technique. This model is trained using not only 90% of the data as for the cross–validation but using all samples available.

4. Multiple analyses of the models are prepared and stored for later retrieval. This includes a detailed comparison of the different models including the data necessary to plot comparison diagrams.

The regression techniques, which are automatically compared by the analysis libraries, can be configured in detail. This means that additional regression techniques can be included

easily. Furthermore the configuration of each of the regression techniques can be provided and is included in the comparison.

By combining the analysis possibilities of R with the functions provided in the Analysis Library, a huge range of analyses can be done. As all the analysis, which was done in this thesis, is automated, it can easily be repeated for verification purposes or to analyze other systems or other configurations.

5.2.3. Related Benchmarking Tools

The automated execution of benchmarks and the automated analysis of their results is not a new problem. This means that tools addressing this task have already been developed. This section explains why a new development approach is taken instead of reusing the existing tools.

Three tools were evaluated for the experimental automation: The *Software Performance Cockpit* [WH11, WHHH10], the *Ginpex* tool [HKHR11] and *Faban* [Fab]. All three tools were designed to help the researcher with the benchmarking process. The decision for writing an own solution and not using and extending one of these tools was made based upon the following facts.

An important required feature is the automated analysis of the benchmark results, the automated model generation and the automated analysis of the regression models. This feature is unsupported by all three tools: While the Software Performance Cockpit contains basic analysis and model generation features, these are designed in an interactive way: The researcher can use the user interface to check the results and generate the models. These steps cannot be automated easily. Ginpex does include the visualization of the results but lacks the ability to execute the analysis and the model generation. Faban does not included any analysis, it only collects the results and saves them to the researchers hard disk.

All three tools lack the integration of any suitable storage benchmark which has proven to be the more complex part in the development comparing to the logic which could have been reused partly from the existing tools. This means that reusing the tools would still have required to write the Benchmark Driver for FFSB. Still it is easily possible to integrate this tool in, e.g., the Software Performance Cockpit.

The Software Performance Cockpit provides a framework for benchmark execution and result analysis. This framework aims at the measurement and modeling of generic software systems. In contrast to this, the Storage Benchmark Harness provides an domain specfic approach focused on the performance storage systems. On the technical side, the Software Performance Cockpit does not support the execution of benchmarks on multiple hosts, especially not in a synchronized way. Only one host can return results whereas the other hosts can only act as a load driver. Collecting the results of the benchmark runs on multiple machines at the same time is an important feature for this thesis. Additionally, the analysis features of the Software Performance Cockpit are tailored towards a fast and interactive analysis by the researcher and not towards a fully automated process. Furthermore, the whole benchmarking and model generation process isn't automated in the Software Performance Cockpit. The Software Performance Cockpit also relies on Java Remote Method Invocations (RMI) for the communication between the hosts and the benchmark tool. This solution requires running Java on the systems under test. This introduces another dependency which might be even impossible to fulfill on some architectures where no suitable Java implementation exists. The Software Performance Cockpit contains multiple features related to the execution of the benchmarks which are not supported by the Storage Benchmark Harness: It supports adaptive benchmarking, where the

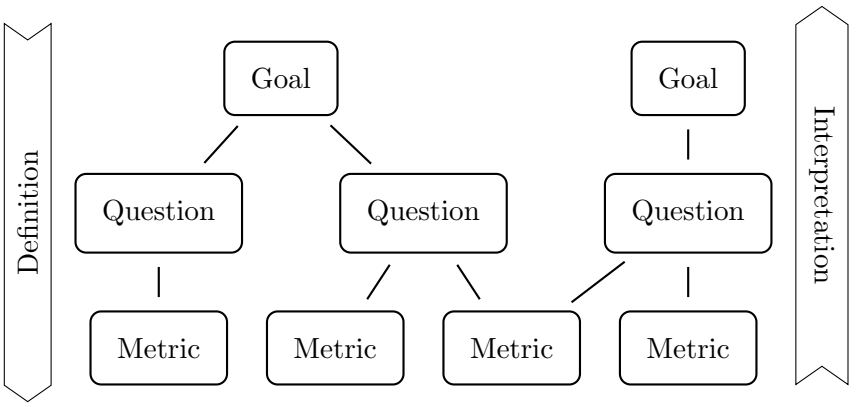


Figure 5.6.: GQM plan schema (based on [BCR94] and [Fab11]).

results are automatically refined to improve the models. Additionally, it supports a much more advanced definition of the configuration and parameters.

The *Ginpex* tool aims in another direction: It helps to generate performance experiments from a model–based specification. This approach was not suitable for this thesis as existing benchmarks had to be reused. On the technical side, Ginpex does not support the collection of the results of external benchmarks. Only simple metrics, like runtime of the whole benchmark and the CPU utilization, are supported. As mentioned above, the collection of the results from the benchmark is one of the required features of the tool. Ginpex does include the visualization of the results but it provides no regression techniques to build prediction models from the collected data. Additionally, Ginpex use RMI in the same way as the Software Performance Cockpit to communicate with the systems under test. Ginpex support a sophisticated configuration where many different experiments can be defined using a high level model. These experiments can involve the benchmarking of CPU, network and storage performance.

The *Faban* benchmark harness is a framework focused on the creation and execution of performance workloads. As with the other tools, it does not include the features necessary for storage benchmarking. On the technical side it does not include a mechanism for running multiple experiments described by a single configuration. As mentioned above, Faban does not contain any benchmark result analysis or regression modeling. It does allow the definition of the benchmark jobs using Java code. This means that the benchmark jobs are not a static specification but instead can contain logic.

5.3. GQM Plan

The analysis of the benchmark results and the regression models is one of the goals of this thesis. Another goal is to make the analysis process as repeatable as possible. To simplify not only the process itself and the interpretation of the results, but also the repeatability, a systematic approach is taken.

The *Goal/Question/Metric Approach* [BCR94] (short GQM) provides a framework which helps to answer research questions in a systematic way. GQM provides two perspectives on the research progress: For the initial analysis it helps to execute the analysis in a methodical top–down way. For the later interpretation it provides a bottom–up approach.

The main component of GQM is the GQM plan. See Figure 5.6 for the overall structure of a GQM plan. It consists of three layers: The "goals" layer defines at a conceptual level the different objectives. These goals can for example be products, processes or resources. The next level is the operational level. It contains one or more questions for each of the

Goal	Question	Section
Part I: Experimental Evaluation and Analysis		Chapter 6
Evaluation of Measurement Setup	How reproducible are the experiments results?	6.1
Evaluation and Analysis of Parameters	Which parameters have an influence on the response time?	6.2
	What is the influence of virtualization?	6.3
Part II: Performance Modeling		Chapter 7
Evaluation and Analysis of Modeling Results	How good is the interpolation of the regression models when using synthetic test sets?	7.1.1
	What interpolation abilities do the regression models show when being tested using newly collected samples?	7.1.2
	How good do the regression models extrapolate when using synthetic test sets?	7.1.3
	How is the extrapolation ability of the regression models when testing using newly collected data?	7.1.4
	How many measurements are needed for an accurate model?	7.1.5
	How can the regression modeling of nominal scale parameters be improved?	7.1.6
Evaluation and Analysis of Regression Techniques	How does the generalization ability of the different regression techniques compare?	7.2.1
	What are the advantages and disadvantages of the modeling techniques?	7.2.2
	Which configuration parameters of the regression techniques can improve the prediction results?	7.2.3

Table 5.4.: The GQM plan for this thesis with references to the matching sections in the following chapters.

goals. These questions characterize how the goal can be achieved. They specify which aspects of the object focused in the goal should be approached. At the lowest level, the quantitative level, each question has one ore more metrics. These metrics are used to quantitatively answer the questions and therefore approach the goal. The metrics must be defined before they are used. Figure 5.6 shows that metrics can be used for two or more questions if necessary. Often a metric gives insight to more than one aspect of the object under research.

The specific GQM plan for this thesis is depicted in Table 5.4. It is grouped into two parts: The first part discusses the results of the benchmarking in detail. It first checks if the measurements setup is suitable. In a next step, the question which parameters have an influence on the storage performance is discussed. The second part uses the results from the first part and creates regression models based on the benchmark results. These regression models are analyzed in-depth and checked for their generalization abilities. Additionally, alternative approaches to benchmarking and the performance model creation are checked. The second goal of this second part is the evaluation of the regression techniques which can be used for generating storage performance models.

6. Experimental Evaluation and Analysis

6.1. How reproducible are the experiments results?

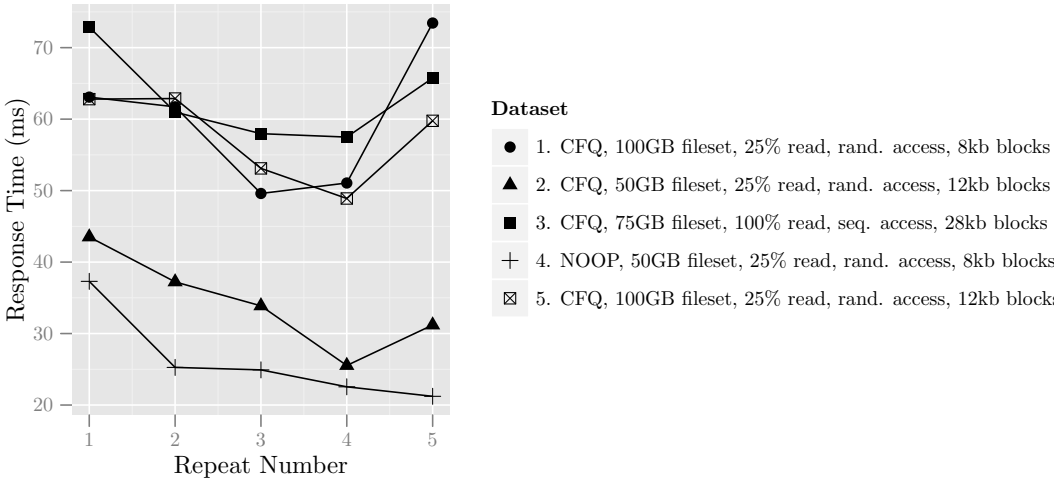
This question is of major importance for the interpretation of the results. As each configuration is first benchmarked with a warm–up run and afterward with five separate benchmark runs, the differences of these repetition can be used to answer this question. See Section 5.1 for a detailed view on the benchmarking methodology.

The two metrics which are used to answer this question are the relative error and the standard deviation of the repetitions. If there is no outer influence and no dependency between the repetitions, the standard deviation and the relative error should be zero. The higher the standard deviation is, the bigger are the difference between the measurements. These two metrics are applied separately to read and write requests. As explained above, each of the 1120 configurations is repeated five times. These repetitions are executed directly after each other. For the calculations in this section, the standard deviation and the relative error of the repetitions mean response time is calculated for each of the 1120 configurations: For mean response times of the five repetitions of one configuration, defined as $R = \{r_1, r_2, r_3, r_4, r_5\}$, the standard deviation σ_R is calculated as explained in Section 4.1.5. The relative error is calculated as follows:

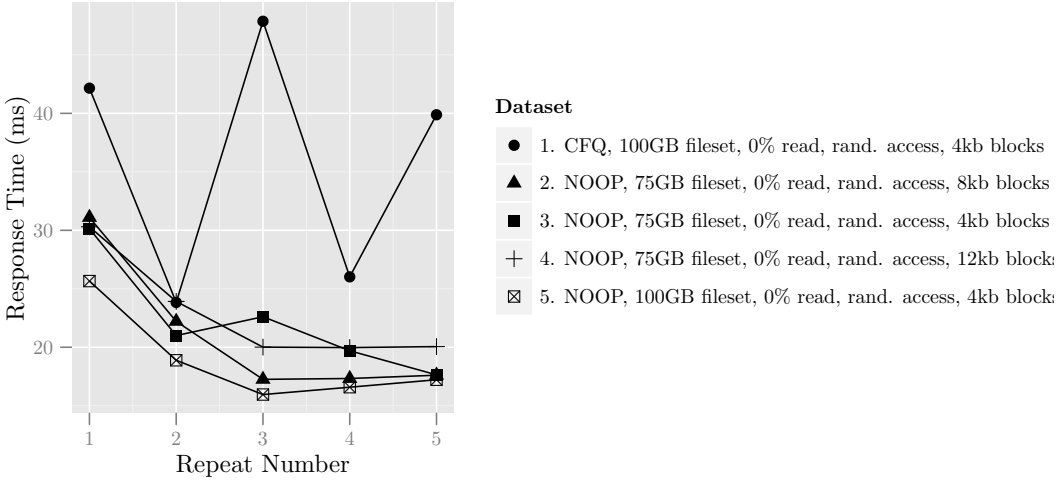
$$\text{rError} = \frac{\sigma_R \cdot 100\%}{\sqrt{5} \cdot \overline{R}}$$

This leads to one value for the standard deviation and one value for the relative error for each of the configurations. These metrics is used to judge on the stability of the single experiments in the following paragraphs.

The experiments with the highest standard deviation in their repetitions are depicted in Figure 6.1. The top five standard deviations for the read requests range from 9.8 ms to 6.23 ms. The CFQ scheduler generates much more unstable measurements than the NOOP scheduler for read requests. Looking at all experiments which were run using the CFQ scheduler, a mean standard deviation of 2.00 ms is calculated. In contrast to that, those experiments which run on a NOOP scheduler only have a mean standard deviation in their measurements of 1.04 ms. This means that the CFQ scheduler roughly doubles the standard deviation compared to the NOOP scheduler. It is difficult to identify other similarities in the experiments with a high standard deviation.

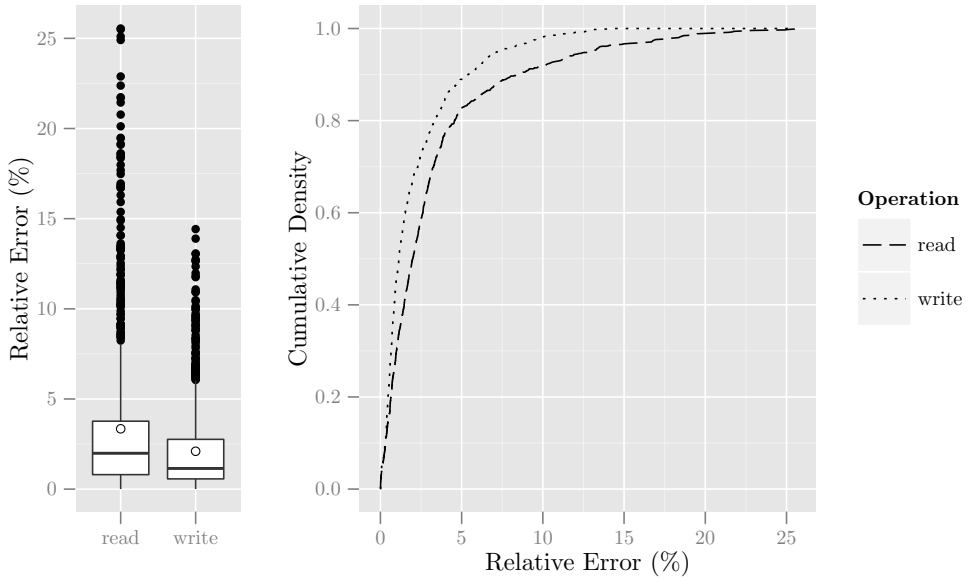


(a) Read Requests



(b) Write Requests

Figure 6.1.: Top 5 experiments with the highest standard deviation in the mean response time of their repeats.



(a) Boxplot (b) CDF

Figure 6.2.: Boxplot and cumulative distribution function visualizing the distribution of the relative error in the repetition of the experiments.

	read (%)	write (%)
lower whisker	0.00309	0.00682
lower quartile	0.80387	0.56788
median	1.98863	1.14751
upper quartile	3.77078	2.76742
upper whisker	8.02942	6.05783

Table 6.1.: The underlying statistical values for the boxplot in Figure 6.2.

For the write requests the situation is easier to analyze: The NOOP scheduler for huge file sets and with pure write workload generates the most unstable results. One exception is the first experiment which can be regarded as an outlier. The other results show a characteristic behavior: The response time reduces over the time. This is a clear sign that the run time of the benchmark runs is not long enough: The runs are not independent from each other. With a warm–up time of two more minutes the results would have been stabilized. Nevertheless, the question remains why the response time decreases with the repeat number increasing. One would expect that the response time increases due to a full write cache in the storage device after some time. An interesting thing to notice is that for write requests the configurations with the highest standard deviation in their repetitions are those, which do pure write workloads.

For the next examinations, the relative error is used as metric. It is calculated by dividing the standard deviation by the square root of the repeat count (five in this case) multiplied by the mean of the repeats. This allows to judge on the deviation of the results in a relative way. Figure 6.2 contains two boxplots and a CDF which show the distribution of the relative errors for both read and write requests. The underlying data can be found in Table 6.1.

For the write requests, the maximum relative error is 14.4% which is an acceptable value, especially under the conditions the measurements were retrieved. Those measurements with a huge relative error within their repetitions show the same characteristics as the worst measurements using the standard deviation: The response time gets better over the first 3 repeats and then stabilizes.

For the read requests the highest relative error is 25.5%. This is a much bigger number which leads to the assumption that read requests are much more unstable than write requests. This assumption can be backed up by the fact that the read cache of the DS8700 is much larger than the write cache. The mean relative error of all measurements does not show this difference: It is 3.34% for read requests and 2.10% for write requests.

To summarize this answer: The repetitions are not as stable as they are desirable. It is not clear why this is the case. Due to the complex nature of the storage system and the unavailability of analysis tools, the cause of these effects remain unknown. A longer measurement time per repetition would increase the stability. Instead of benchmarking for five times one minute it would be necessary to benchmark five times five minutes. Additionally an increase in the warm–up time is necessary, especially for the write requests. This was unfeasible due to time constraints: This increase of the benchmarking time would also increase the total benchmarking duration. With a benchmarking time of already more than one week a further increase is not feasible. It is still possible to use the current results as if only a single run of five minutes was run. This can be done easily by calculating the mean of all five repetitions and treating this value like a single repeat. Nevertheless, the repetitions are needed for the analysis of the parameters influence in the next section.

6.2. Which parameters have an influence on the response time?

The answers to this question plays an important role on the choice on the parameters which should be benchmarked in detail. Additionally, the answer to this question helps to decide which parameters should be included in the models later in Chapter 7. If the answer to this question concludes that some parameters have no influence on the storage performance, these parameters could be left out of the model generation and further benchmarking.

An ANOVA of the parameters is the typical way to solve this problem and the results of the ANOVA are used as metric to answer this question. Section 4.2 contains an explanation how to interpret the results of an ANOVA analysis. To execute this ANOVA, the mean of each benchmark run was calculated separately for read and write requests. For both, the read and write requests, two separate ANOVAs were executed. Table 6.2 show the results of these analyses separately for read and write requests. As the probability that each parameter has no influence on the result is nearly zero (the exact value is smaller than 2^{-16}) for all parameters, it can be assumed that all five parameters influence the variation of the mean response time. This is true for both, read and write requests. Figure 6.3 contains a graphical representation for the quantification of the influence of the parameters on the variation. This quantification show a huge impact of the scheduler. This is a logical result as the CFQ scheduler is somehow unsuited for the IBM DS8700 with its sophisticated caching mechanisms. Another point is the huge relative influence of the residuals. These residuals can either result from measurement errors or can simply be those parts of the variation which cannot be explained using the supplied parameters. Later in this section it becomes clear that the second case is true: The measurement error does in fact not have such a huge influence on the variation. In this context this only means that only about 55% of the variation can be explained using the five parameters and all the other variation is counted towards the residuals. An interesting difference between the read and the write requests is the influence of sequential and random access. For read requests this does not have a huge influence while for write requests changing from sequential to random access affects the response time.

In the following paragraph, it is checked that not only the simple parameters influence the response time but also their interactions. To verify this assumption, an ANOVA test

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
filesetSize	4	129805.09	9.36	32451.27	243.35	0.0000
blockSize	7	67227.21	4.85	9603.89	72.02	0.0000
sequentialAccess	1	64252.95	4.63	64252.95	481.83	0.0000
scheduler	1	396445.48	28.58	396445.48	2972.94	0.0000
readPercentage	5	91672.27	6.61	18334.45	137.49	0.0000
Residuals	4781	637552.54	45.97	133.35		

(a) Read Requests

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
filesetSize	4	55241.76	5.58	13810.44	181.98	0.0000
blockSize	7	84777.97	8.56	12111.14	159.58	0.0000
sequentialAccess	1	118402.39	11.96	118402.39	1560.15	0.0000
scheduler	1	338859.54	34.22	338859.54	4465.03	0.0000
readPercentage	5	30199.64	3.05	6039.93	79.59	0.0000
Residuals	4781	362838.72	36.64	75.89		

(b) Write Requests

Table 6.2.: Multidimensional ANOVA test on all parameters without interactions.

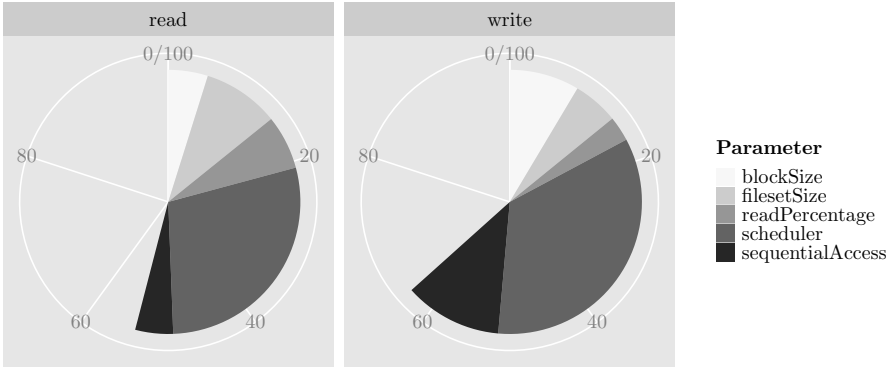


Figure 6.3.: Pie chart of the amount of influence on the variation of the response time per parameter. Missing values to 100% are the influence of the residuals.

is used on both, the parameters and the interaction between two parameters. Table 6.3 show the results, again split into read and write requests. For both request types nearly all parameters and their interactions seem to influence the response time. For read requests, exceptions are the interaction term between the block size and both, the file set size and the read percentage. It should be noted that all three parameters have an influence as standalone terms, there is just no combined influence. For write requests no clear statement can be made: The question if the two interaction terms mentioned before also do not influence write requests is not clear. From the ANOVA one can at least tell that it's much likelier that both interactions in fact play a role than the opposite.

For the quantitative discussion the first important point is the reduction of the influence of the error terms. Figure 6.4 contains two pie charts resembling the influence of the different factors. To increase the readability, parameters with a small influence have been merged to "other". The influence of the residuals is reduced from about 46% to 7% for read requests. This results from the fact that more variation can be explained using the interactions and therefore less variation must be explained by the residuals. Additionally to the scheduler, which still plays an important role, there is a huge interaction between the sequential access flag and the scheduler for both, read and write requests. It leads to the conclusion that for one of the two schedulers the changing from sequential to random access or vice versa has a huger influence than for the other scheduler. Most of the other interactions do not play a very important role, especially for write requests where all relative influences

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
filesetSize	4	129805.09	9.36	32451.27	1470.20	0.0000
blockSize	7	67227.21	4.85	9603.89	435.10	0.0000
sequentialAccess	1	64252.95	4.63	64252.95	2910.98	0.0000
scheduler	1	396445.48	28.58	396445.48	17960.94	0.0000
readPercentage	5	91672.27	6.61	18334.45	830.64	0.0000
filesetSize:blockSize	28	300.09	0.02	10.72	0.49	0.9897
filesetSize:sequentialAccess	4	48776.42	3.52	12194.10	552.45	0.0000
filesetSize:scheduler	4	7805.86	0.56	1951.47	88.41	0.0000
filesetSize:readPercentage	20	4569.28	0.33	228.46	10.35	0.0000
blockSize:sequentialAccess	7	43219.46	3.12	6174.21	279.72	0.0000
blockSize:scheduler	7	15613.25	1.13	2230.46	101.05	0.0000
blockSize:readPercentage	35	608.33	0.04	17.38	0.79	0.8100
sequentialAccess:scheduler	1	388654.37	28.02	388654.37	17607.97	0.0000
sequentialAccess:readPercentage	5	23591.17	1.70	4718.23	213.76	0.0000
scheduler:readPercentage	5	1445.42	0.10	289.08	13.10	0.0000
Residuals	4665	102968.87	7.42	22.07		

(a) Read Requests

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
filesetSize	4	55241.76	5.58	13810.44	1743.72	0.0000
blockSize	7	84777.97	8.56	12111.14	1529.17	0.0000
sequentialAccess	1	118402.39	11.96	118402.39	14949.62	0.0000
scheduler	1	338859.54	34.22	338859.54	42784.78	0.0000
readPercentage	5	30199.64	3.05	6039.93	762.61	0.0000
filesetSize:blockSize	28	288.46	0.03	10.30	1.30	0.1332
filesetSize:sequentialAccess	4	8163.24	0.82	2040.81	257.67	0.0000
filesetSize:scheduler	4	11948.33	1.21	2987.08	377.15	0.0000
filesetSize:readPercentage	20	1232.97	0.12	61.65	7.78	0.0000
blockSize:sequentialAccess	7	18968.06	1.92	2709.72	342.13	0.0000
blockSize:scheduler	7	13985.36	1.41	1997.91	252.26	0.0000
blockSize:readPercentage	35	553.49	0.06	15.81	2.00	0.0004
sequentialAccess:scheduler	1	263456.40	26.60	263456.40	33264.30	0.0000
sequentialAccess:readPercentage	5	6396.13	0.65	1279.23	161.52	0.0000
scheduler:readPercentage	5	899.03	0.09	179.81	22.70	0.0000
Residuals	4665	36947.25	3.73	7.92		

(b) Write Requests

Table 6.3.: Multidimensional ANOVA test on all parameters including interaction terms between two parameters.

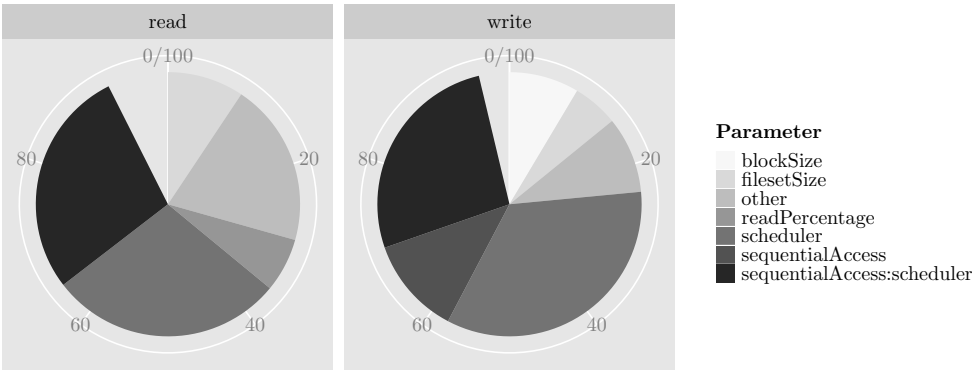


Figure 6.4.: Pie chart of the amount of influence on the variation of the response time per parameter. It includes interactions between two parameters. Values smaller than 5% have been aggregated to "other". Missing values to 100% are the influence of the residuals.

are below 4%.

Including a higher level of interaction in the ANOVA has not proven to be of much value: None of the interaction terms has more than 3% influence on the variation. Additionally, the ANOVA table gets large and more difficult to read. Nevertheless, it should be noted that the amount of variation that cannot be explained using the parameters and their interactions goes down to 1.29% for read requests and 0.47% for write request. The results from the two ANOVAs can be found in Appendix A.

The input for these four ANOVAs are the 1120 configurations which were repeatedly benchmarked five times. This leads to a total of 5600 measurements which are analyzed. A possiblity to reduce the number of measurements required is to limit each parameter only to its extremes. This means that for each parameter only two values are benchmarked, its minimum and its maximum. This is called a $n2^m$ ANOVA [Kou11] because only $n2^m$ measurements are needed where n is the number of repetitions and m the number of parameters. In the case of this thesis this would lead to 160 measurements, only about 3 percent of the measurements needed for a full exploration. The question arises if these reduced sets is still sufficient to check which parameters have an influence on the response time. To analyze this, a subset of the existing measurements is built containing only the extreme values. This means that the following values are included:

- *Block size*: The two extreme values 4 kB and 32 kB are included.
- *File set size*: Again the minimum and the maximum of 1 GB and 100 GB are the only file set sizes which are included in the subset.
- *Read percentage*: For the read percentage it is not possible to simply select the lowest and the highest read percentage to be included in the subset. The reason for this is that a read percentage of 0% would lead to no read request issued and therefore no result would be recorded. The same is true for a read percentage of 100% and write requests. For this reason the values included for the read percentage are 25% and 75%, the second lowest and second highest values which were benchmarked. For a future benchmarks, values closer to the minimum and maximum might lead to better results.
- *Scheduler* and *sequential access* are left unmodified as they already include only two values.

Table 6.4 contains the results of two separate ANOVAs executed on the reduced sample set specified above. Again a separate ANOVA is done for read and write requests. By reducing the samples, the results of the ANOVA got much more insecure. For multiple interaction terms the answer to the question whether they influence the mean response time or not cannot be answered with a high reliability. For example the interaction term between file size and scheduler has a five percent probability to have no influence on the response time. In contrast to that, in the full factorial ANOVA it can be assumed with a very high probability that there is in fact an influence. On the other side for example the interaction term between file set size and block size also has a probability of five percent to have no influence. As stated above this is in fact true, the full ANOVA has a probability of 99% for this term. These two examples show that a five percent probability of influence for a $n2^m$ ANOVA can result in a very high probability and a very low probability in a full factorial ANOVA. Therefore the reduction of the parameter values does not pay out. The quantitative analysis shows that the values are a rough estimate for the values of a full factorial ANOVA. The top four influencing factor are the same for both methods. After this the results diverge with makes an analysis of the less important factors impossible.

The overall answer to this question is: All of the five evaluated parameters and nearly all interactions between two parameters have an influence on the response time. The

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
filesetSize	1	9256.13	14.82	9256.13	201.21	0.0000
blockSize	1	5690.98	9.11	5690.98	123.71	0.0000
sequentialAccess	1	468.83	0.75	468.83	10.19	0.0017
scheduler	1	12468.30	19.96	12468.30	271.04	0.0000
readPercentage	1	5585.59	8.94	5585.59	121.42	0.0000
filesetSize:blockSize	1	17.19	0.03	17.19	0.37	0.5420
filesetSize:sequentialAccess	1	5983.55	9.58	5983.55	130.07	0.0000
filesetSize:scheduler	1	177.83	0.28	177.83	3.87	0.0512
filesetSize:readPercentage	1	183.77	0.29	183.77	3.99	0.0475
blockSize:sequentialAccess	1	2581.49	4.13	2581.49	56.12	0.0000
blockSize:scheduler	1	863.28	1.38	863.28	18.77	0.0000
blockSize:readPercentage	1	201.61	0.32	201.61	4.38	0.0381
sequentialAccess:scheduler	1	11618.73	18.60	11618.73	252.57	0.0000
sequentialAccess:readPercentage	1	526.32	0.84	526.32	11.44	0.0009
scheduler:readPercentage	1	228.67	0.37	228.67	4.97	0.0273
Residuals	144	6624.19	10.60	46.00		

(a) Read Requests

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
filesetSize	1	4020.13	12.12	4020.13	305.96	0.0000
blockSize	1	5704.78	17.19	5704.78	434.17	0.0000
sequentialAccess	1	2650.29	7.99	2650.29	201.71	0.0000
scheduler	1	8790.42	26.49	8790.42	669.01	0.0000
readPercentage	1	311.58	0.94	311.58	23.71	0.0000
filesetSize:blockSize	1	51.27	0.15	51.27	3.90	0.0501
filesetSize:sequentialAccess	1	114.79	0.35	114.79	8.74	0.0036
filesetSize:scheduler	1	854.98	2.58	854.98	65.07	0.0000
filesetSize:readPercentage	1	45.56	0.14	45.56	3.47	0.0646
blockSize:sequentialAccess	1	875.52	2.64	875.52	66.63	0.0000
blockSize:scheduler	1	986.61	2.97	986.61	75.09	0.0000
blockSize:readPercentage	1	22.60	0.07	22.60	1.72	0.1918
sequentialAccess:scheduler	1	6843.28	20.62	6843.28	520.82	0.0000
sequentialAccess:readPercentage	1	1.63	0.00	1.63	0.12	0.7252
scheduler:readPercentage	1	16.60	0.05	16.60	1.26	0.2629
Residuals	144	1892.07	5.70	13.14		

(b) Write Requests

Table 6.4.: Multidimensional ANOVA test on all parameters including interactions between two parameters. Only two values were included for each parameter, leading to a $n2^m$ ANOVA and a reduction of the required benchmark runs.

	read (%)	write (%)
lower whisker	0.00071	0.00485
lower quartile	4.27833	5.05743
median	8.53218	10.41316
upper quartile	14.45540	26.06969
upper whisker	29.14669	51.32413
mean	10.60307	16.67317

Table 6.5.: Underlying data for the boxplot in Figure 6.5.

reduction of the values when going from full factorial to a $n2^m$ experimental design cannot be supported due to ambiguous results.

6.3. What is the influence of virtualization?

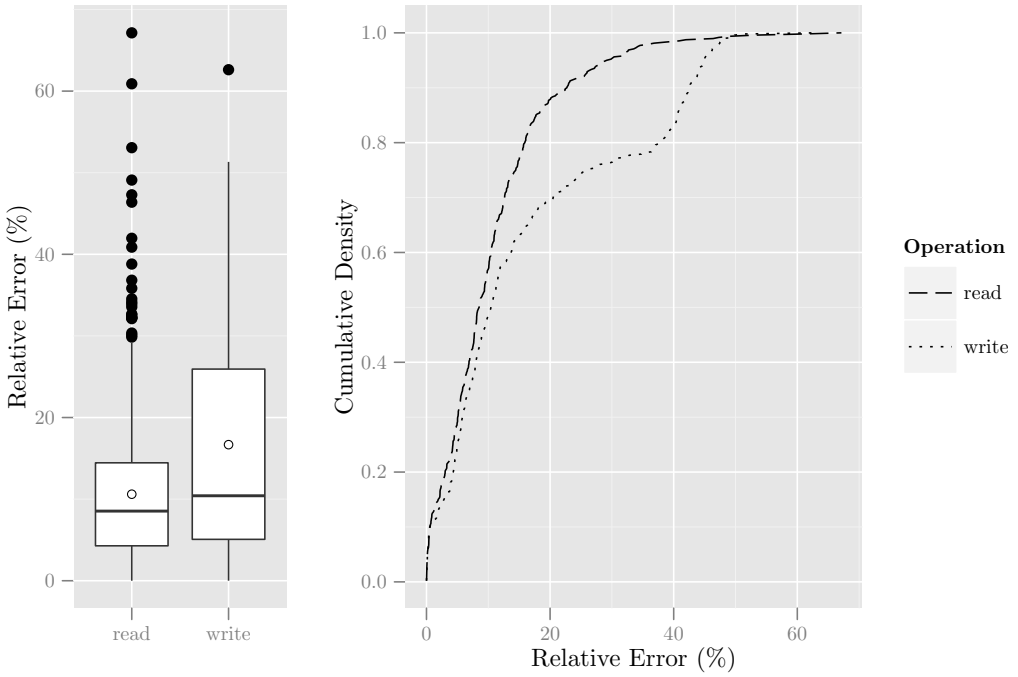
As the IBM System z is a virtualized machine and as virtualization has become an important technology, the question arises if the influence of virtualization can be captured by the measurements. To check this, the measurements, which were gathered previously, are compared to newly run benchmarks with the same I/O load distributed on two virtual machines. The comparison was done using the relative error as metric.

As the NOOP scheduler shows a much more stable results, this examination is limited to this scheduler. The CFQ scheduler is left out intentionally. The 560 remaining configurations explained above, are all benchmarked using 100 threads. To check if the benchmarking process can be used even if multiple virtual machines are involved, new benchmark configurations are created: 560 new configurations are defined using only 50 threads, exactly half of the original 100 threads. Additionally the file set is also halved which means that the following file set sizes are benchmarked: 500 MB, 12.5 GB, 25 GB, 37.5 GB, 50 GB. For each of these 560 configurations two identical benchmark runs are started simultaneously on two virtual machines. The virtual machines are configured identically to the machine where the original benchmarks were run. The simultaneous execution of the benchmark is fully automated using the Storage Benchmark Harness which was developed for this thesis.

By using this setup, the load which is generated by the benchmarks should be identical in the one virtual machine case and the two virtual machine case. Three results exists per configuration: The result from the original run on a single virtual machine and the results from the two virtual machines when running in parallel. Under ideal conditions, one can expect that the mean response times of all three benchmark runs are identical. To check if this assumption is true, the relative error is calculated as following:

$$\text{rError} = \left| \frac{\frac{\text{dualVM1}+\text{dualVM2}}{2} - \text{singleVM}}{\text{singleVM}} \right|$$

In this definition, "singleVM" is the mean response time of the requests when running on a single virtual machine. These are the results from the original measurements. "dualVM1" and "dualVM2" are the mean response time gathered when the benchmark was run on two virtual machines simultaneously. This definition means that the mean of the results of the dual virtual machine case is compared to the single virtual machine case using the relative error. The "singleVM" variable contains the results from the configuration where the thread count and the file set size was doubled comparing to the configuration which was used when obtaining the "dualVM1" and "dualVM2" data. The read and the write requests are aggregated separately because of their different behavior.



(a) Boxplot (b) CDF

Figure 6.5.: Boxplots and CDF for the relative errors of the read and the write requests. The relative error between the case when running the benchmark on two separate virtual machines and the case when running a single benchmark on a single virtual machine is depicted here.

Figure 6.5 shows the results of this analysis: The relative errors of all configurations was included in the two boxplots and in the CFD: One boxplots contains the relative errors of the read requests and one boxplot contains the relative errors of the write requests. The mean is 10.6% for read requests and 16.6% for write requests. This can be regarded as good values, although these number suggest that the assumption made above cannot be fully supported. When looking at the boxplot more detailed, a different behavior between read and write requests can be shown: For the read requests, 50% of the configurations have a relative error in the quite narrow range between 4.2% to 14.4%. This also means that the outliers, which have their maximum at 67.1%, contribute a lot to the high mean relative error. When looking at the CDF, it becomes clear, that about 80% of the samples have a relative error below 18%. Later in this section, the reasons for this behavior are analyzed.

When looking at write requests, the 50% interval is between 5.0% and 26.0%. This means that, compared to the read requests, the majority of the configurations have a huge variations in their relative errors. Based on the numbers, one can conclude that the assumption made above does only hold in a restricted way for write requests.

The question where these high relative errors results from is discussed next. To analyze this, the configurations are split according to their read percentage. The results can be found in Figure 6.6. For the read requests, depicted in the upper part of the diagram, the 0% read percentage has no boxplot. The reason for this is, that with a read percentage of 0%, no read requests were issued and therefore no relative error could be calculated. For the same reason there is no boxplot for the 100% read percentage and write requests.

When looking at the read requests, it becomes clear that for pure workloads, the relative error is very low. Pure workloads are workloads with only read or only write requests. For

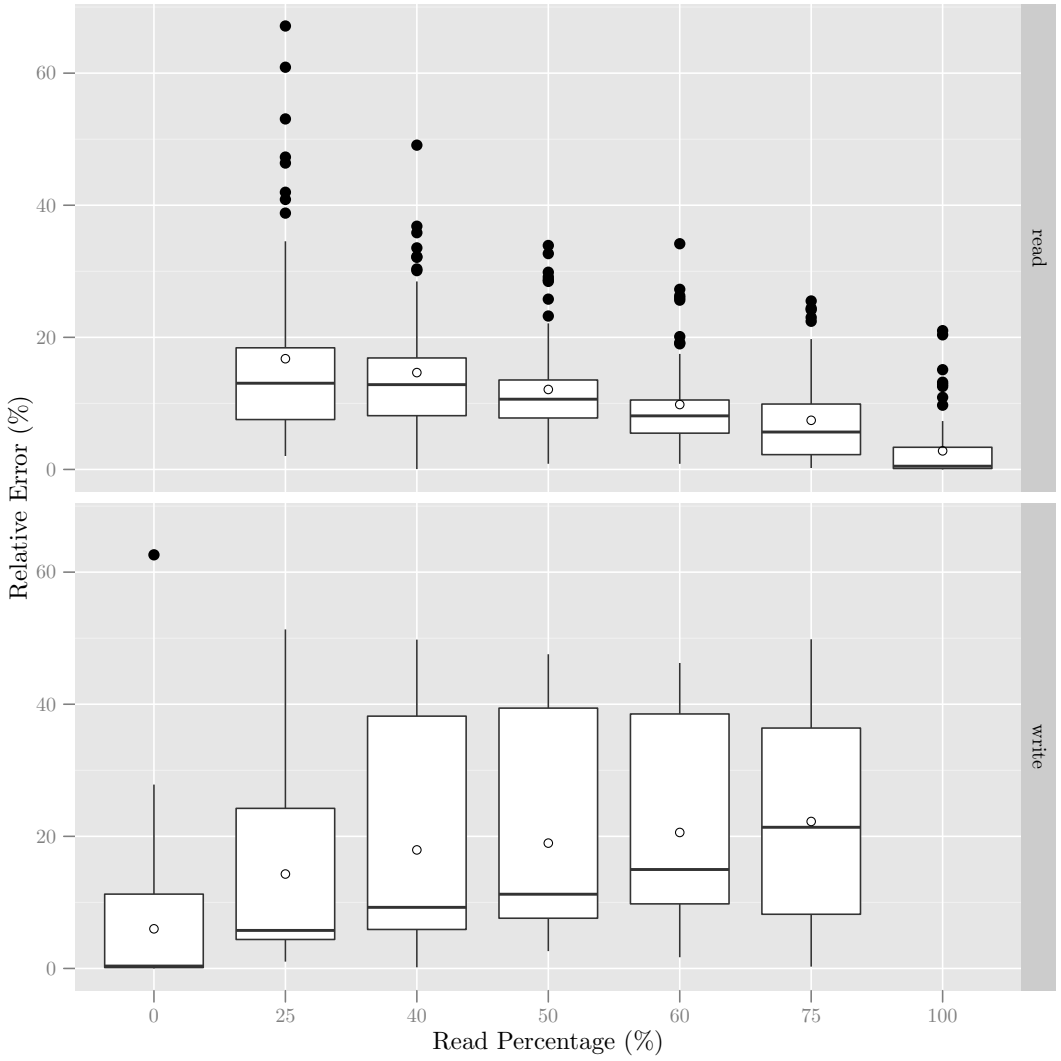


Figure 6.6.: Boxplots comparing the relative errors depending on the read percentage. The read percentage is the amount of operations which are read requests.

the read requests, the mean relative error for pure read workloads is 2.8% which is a very good value. When decreasing the read percentage and therefore increasing the amount of write requests issued, the mean relative error and its variation increase significantly.

For write requests, the relative error is also quite low for a pure write workload: The mean relative error for the configuration with a read percentage of 0% is 6.0% and the variation is quite small. In contrast to the read requests the error and especially the variation goes up very fast when increasing the read percentage.

Although the configurations were chosen to produce the same I/O load on the system in total, the results are not clear: It can be concluded from the numbers and figures presented above that this assumption can only be fully supported for pure workloads. For all other workloads, especially for write requests, the relative error is too high.

7. Performance Modeling

7.1. Evaluation and Analysis of Modeling Results

One of the major tasks of this thesis is the generation of storage performance prediction models. As the generation is fully automated, the question of the quality of the models can be discussed in detail. The following sections discuss the question of interpolation and extrapolation of the regression models.

The models, whose quality is analyzed in the following sections, are generated based on the 1120 configurations defined in Section 5.1. Each of these configurations is benchmarked five times. The mean of the mean response time of all repetitions is recorded for each configuration. This leads to a total of 1120 samples for the regression model creation. The repetitions are not used separately but aggregated because of the short benchmarking time and huge differences in the repetitions as discussed in Section 6.1. If longer measurements are available it would be possible to include these repetitions in the modeling process. Two separate models are generated for read and write requests. This is done because read and write requests have a completely different behavior and different characteristics.

Four different regression techniques are used to generate nine regression models. An unique identifier is given to each of these nine models for referencing purposes.

- *Linear Regression* (as explained in Section 4.3.1):
  - `lm`: A linear model is trained using only the five input parameters. Multiple authors state that this regression technique is inappropriate for modeling storage performance. For this reason no sufficient model quality can be expected. The model is nevertheless included for comparison reasons and to prove the assumptions which have been made during the discussion of the influence of the parameters in Section 6.2.
  - `lm_2param_inter`: This linear model contains the five input parameters and also the ten interaction terms between pairs of two parameters. This leads to a total of 15 coefficients which have to be computed. This enables the model to adapt to interactions between two parameters.
  - `lm_3param_inter`: This model additionally contains the interaction terms between three of the parameters. There are 10 of these terms. This leads to a total number of $5 + 10 + 10 = 25$ terms in the model. This setting allows the model to additionally include interactions between three parameters.

  - `lm_4param_inter`: Similar to above, this model contains the interactions between up to four parameters. There are five of these interaction terms which leads to a total number of $5+10+10+5 = 30$ terms in the model. The inclusion of the interactions allows the model to even model interactions between four parameters.
  - `lm_5param_inter`: This model includes everything of the above model plus a single term modeling the interactions between all five parameters. For this reason the total number of terms is $5 + 10 + 10 + 5 + 1 = 31$ terms. This is the highest number of terms possible for five parameters.

- *Multivariate Regression Splines* (as explained in Section 4.3.2 for the statistical background):
  - `mars`: This MARS model was trained allowing no interaction terms. This means that the *degree* parameter of the MARS algorithm was set to 1. This makes this model comparable to the `lm` model above which also does not contain interaction terms.
  - `mars_multi`: For this MARS model the degree parameter was set to a value of five. This extension allows the MARS algorithm to choose interaction terms between up to five parameters. This extends the list of possible terms but does not mean that these term are necessarily included in the final model. This setting allows the MARS model to include the same interaction terms as the `lm_5param_inter` linear regression model. For the MARS algorithm this setting allows to choose the actual terms for all interaction terms, including those using two, three, four, and five parameters.

- *Regression Tree* (as explained in Section 4.3.3 for background information) `cart`: A regression tree is trained on the data using the cart algorithm.
- *M5* (as explained in Section 4.3.4 for detailed explanation) `m5`: An M5 regression tree with attached linear models trained using the training samples.

All other configuration parameters of the regression techniques are set to their implementation defaults. In Section 7.2.3 these configuration parameters are further examined.

All regression techniques except CART are not designed for non interval scales. Some of the parameters discussed in this thesis are on a nominal scale. This includes the scheduler and the sequential/random access flag. To model these nominal parameters, a classification model would be a better choice. The problem here is that classification models do not support interval scales. For this reason the nominal parameters are transformed to an interval scale by assigning integer values to each level. For the sequential/random access flag a new parameter "sequentialAccess" is created. It has the value "0" for sequential access and "1" for random access. The scheduler parameter is transformed into a "schedulerNOOP" parameter. This parameter is "1" if the scheduler is NOOP and "0" if the scheduler is CFQ. This approach can be extended to more values, for example if a third scheduler had been benchmarked.

The quality of the models can only be judged using samples which were not used in the training process (see Section 4.5 for the reasons). The samples which were not used are called the training set. By calculating the predictions of the models for this training set and then comparing the predicted with the actual values the model quality can be judged. For this process different metrics are used: The root mean square error (RMSE) and the medium absolute percentage error (MAPE). For their differences see Section 4.4. By using these metrics, the quality of the models is an indication for the generalization ability. The

Model	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
lm	11.51	87.02	8.77	66.32
lm_2param_inter	4.80	31.40	3.26	18.12
lm_3param_inter	3.13	16.72	2.62	10.35
lm_4param_inter	2.95	13.43	2.56	9.26
lm_5param_inter	2.96	13.47	2.56	9.19
cart	6.16	38.40	5.20	38.11
mars	11.54	86.36	8.75	66.84
mars_multi	4.86	34.34	2.86	16.00
m5	1.69	6.49	0.81	4.24

Table 7.1.: Underlying data for the boxplots in Figure 7.1, which show the comparison of the interpolation abilities of the regression models when tested using 10-fold cross–validation.

generalization ability is split into two parts: The generalization ability within the modeled ranges, the so–called interpolation and out of the ranges, the so–called extrapolation.

The results of the following sections are later summarized in Section 7.1.7.

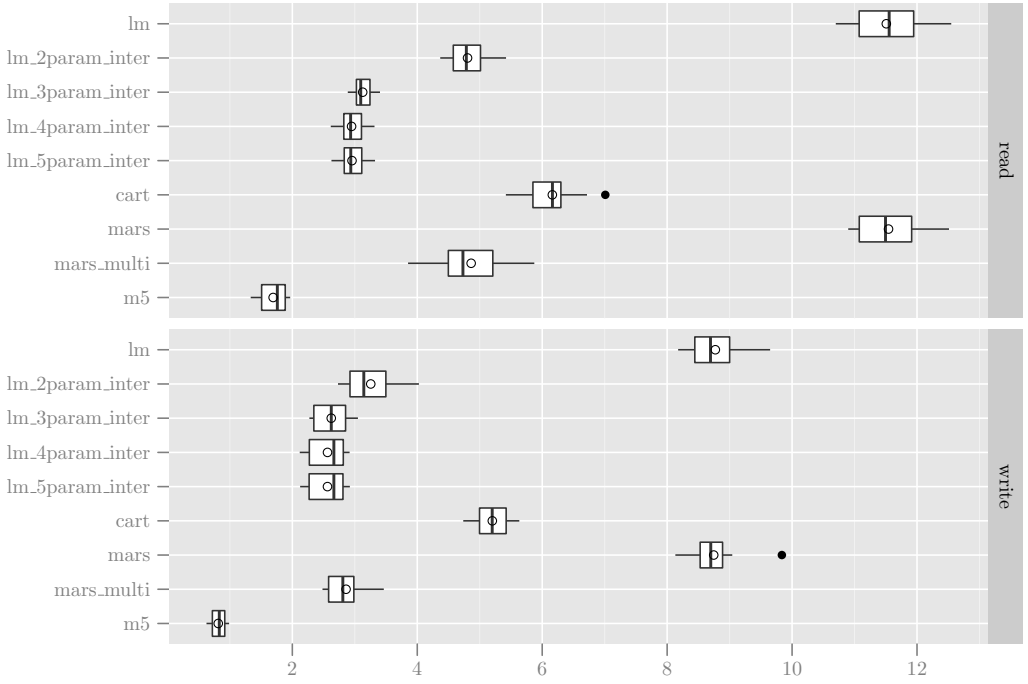
7.1.1. How good is the interpolation of the regression models when using synthetic test sets?

To check how good the regression models interpolate, the 1120 input samples are divided into two groups: 90% of the data is used to train the model and 10% of the data is used to check how good the model generalizes. Ten of these training–test pairs are generated. Then each model specified above is trained using the ten training sets and tested using the ten test sets. RMSE and MAPE are used as metrics. This leads to ten values for both model metrics for each of the nine models. This process assures, that for comparability reasons, each model is trained using exactly the same training data. Although the goals seem to be different, this approach is the same as 10-fold cross–validation (see Section 4.5 for an explanation). This process is needed because the naive approach, to build a model using the whole data set and then take some test samples out of this set to check the models quality, is not an acceptable way. As explained before, this process would not detect an overfitted model.

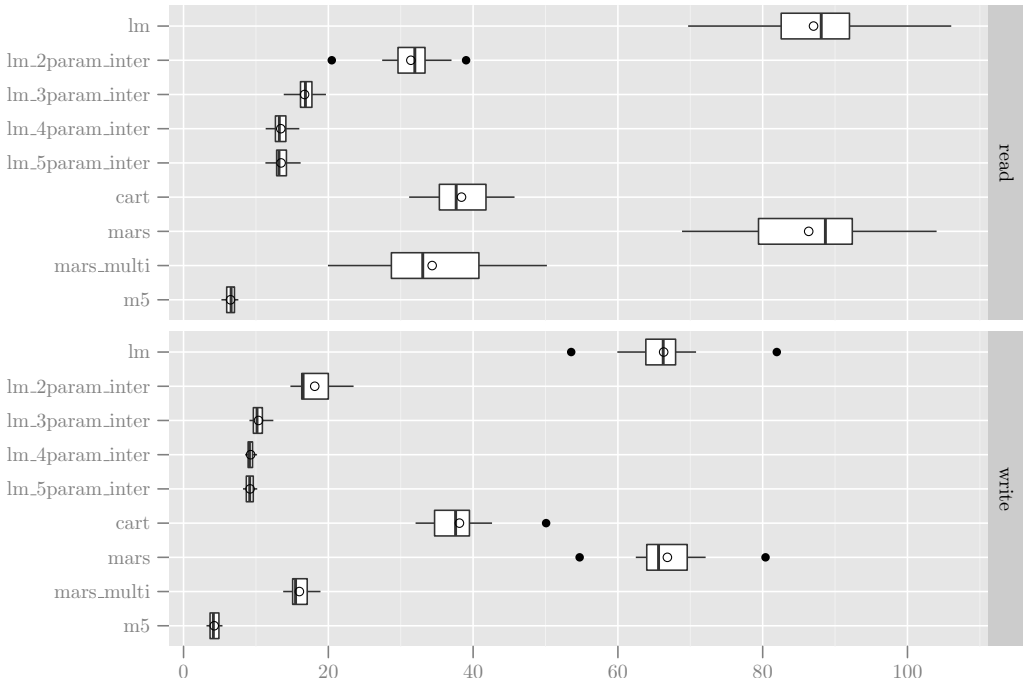
Figure 7.1 contains boxplots for each model and each metric. As there were separate models for read and write operations, two boxplots exist for each model and each metric. Both metrics produce smaller values if the model is better. Each boxplot includes the results of the ten validation runs. When comparing the models, the mean of the ten cross–validation runs is used. The mean is depicted as a white dot in the box plots. Table 7.1 contains the underlying data.

Although the definitions of the metrics differ, they lead to the same order in the models. Both metrics clearly identify the M5 model as the model with the best generalization ability. The M5 model (`m5`) performs very well: For the read model, the mean absolute percentage error (MAPE) is 6.49% and 4.24% for the write model. This means that the average prediction error is 6.49% or 4.24%. The root mean square error (RMSE) is 2.96 ms for the read model and 0.81 ms for the write model. These results are very good, especially when considering the complexity of the system. The M5 model is able to adapt very well to the test data and generalize.

The second best model is the linear regression model which includes all interactions up to five parameters (`lm_5param_inter`): With a MAPE of 13.47% and 9.19% for read and write models and an RMSE of 2.96 ms and 2.56 ms respectively, it performs significantly



(a) RMSE in ms



(b) MAPE in %

Figure 7.1.: Comparison of the quality of models generated using the full data set. Two different metrics were used: Root Mean Square Error (RMSE) and Medium Absolute Percentage error (MAPE).

worse than the M5 model. Nevertheless these results are also very good, especially when keeping the simple nature of the linear regression models in mind. It is interesting to see that a linear regression is sufficient for adapting to the rough structure of the storage performance data.

The inclusion of the single interaction term which consists of all parameters in the `lm_5param_inter` model does not provide a significant improvement. The `lm_4param_inter` model which does not include this term has only a 0.04% (read model) or 0.07% (write model) lower MAPE and a 0.01 ms (read model) or 0.003 ms (write model) lower RMSE. The differences are so small that one can conclude that the interaction between all five parameters does not play a role when predicting the storage performance.

It might seem surprising that more sophisticated techniques like MARS or CART did not perform better: The MAPE of the multidimensional MARS model, which can include the same interaction terms as the `lm_5param_inter`, has a MAPE of 34.34% and 16.00% which is about the double value of the linear model and more than four times the value of the M5 model. The values them–selves are not very good but could be acceptable, depending on the use case of the models. They can be used for a rough estimate. The reason for this behavior is discussed later in Section 7.2.1 where the modeling techniques are compared.

When comparing read and write models, the write model are better for all modeling techniques. This results from the benchmark results: They are more stable and easier to predict for write requests. The reason for this behavior is mainly the smaller cache for write requests compared to the cache for read requests: The write cache is only 2 GB in size, whereas the read cache is 50 GB large.

In general the models perform very good. It must be kept in mind that a complex storage system with many layers is modeled. A MAPE of 6.49% is acceptable for most uses of the model. Later in Section 7.2.2 a detailed comparison between the different model techniques is done.

7.1.2. What interpolation abilities do the regression models show when being tested using newly collected samples?

While the previous question analyzes the interpolation abilities of the regression models using a synthetic test set generated from the existing samples, this section takes another approach: It uses newly collected benchmark results.

For this section the whole 1120 samples are used to train the nine models. This is a slight increase in the training samples: In the previous section, the models are trained using 1108 samples (90% of 1120), this time the whole 1120 samples are used. To verify the interpolation abilities of these models, new samples are collected. The configurations of these samples are composed as follows:

- The *block size* is randomly selected from the range 4 kB to 32 kB.
- The *read percentage* is randomly selected from the range 0% to 100%.
- The *file set size* is taken randomly from the range of 1 GB to 100 GB.
- The *sequential access flag* is randomly chosen between random access and sequential access.
- The *scheduler* is randomly chosen out of the two available schedulers NOOP and CFQ.

All other benchmark settings remain at the same values which are used to obtain the original results. As the FFSB benchmark only supports block sizes which are a multiple

Model	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
lm	13.83	79.34	9.25	62.28
lm_2param_inter	6.20	25.04	3.05	16.96
lm_3param_inter	4.68	15.60	2.57	10.56
lm_4param_inter	4.81	13.85	2.61	10.05
lm_5param_inter	4.81	13.87	2.60	10.01
cart	9.19	35.28	4.68	33.97
mars	13.19	79.49	9.23	64.65
mars_multi	6.49	28.52	2.84	16.64
m5	3.56	9.27	2.44	10.39

Table 7.2.: Comparison of the interpolation abilities when using 200 newly collected random samples. This is the underlying data for the bar plots in Figure 7.2.

of 512 bytes the block sizes are rounded to the nearest 512 byte step. Because the file set is composed out of files with a size of 16 MB, the file set size is rounded to the nearest 16 MB step. All other values are left at their randomly selected values. This means that file set size and read percentage can take any value in the mentioned range. No rounding is done.

This approach randomly selects configuration within the benchmarked range to test the interpolation of the models. 200 configurations are randomly composed out of the specification above. Each of these configurations is benchmarked in five runs each lasting for one minute. The results of the five runs are averaged. The approach of benchmarking new samples can be regarded as more realistic compared to the previous section: It simulates the later use of the models where other values than those benchmarked are predicted using the models.

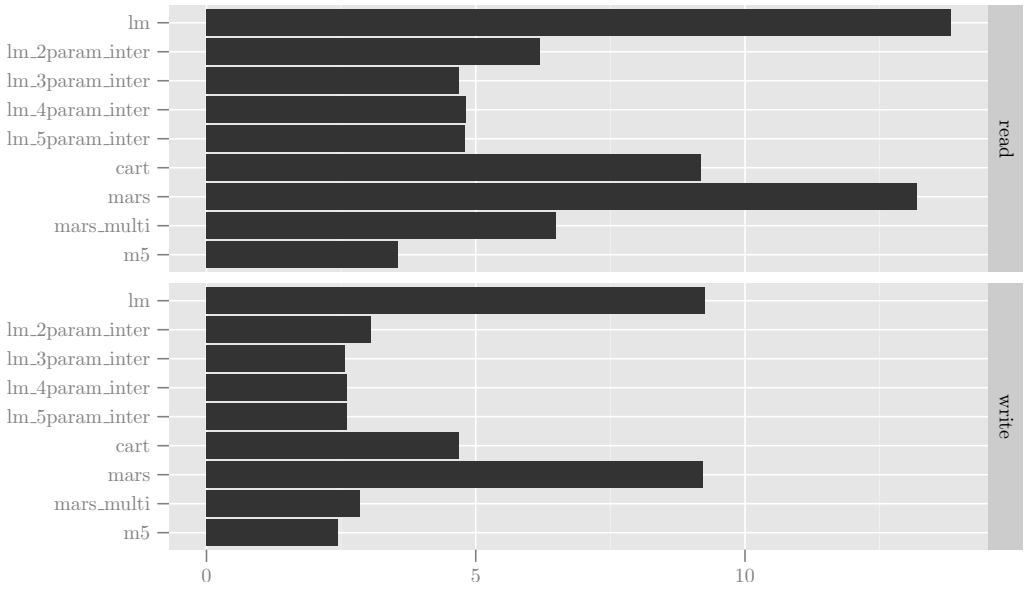
Figure 7.2 and Table 7.2 contain the results for this interpolation test. The results are very promising: The samples are collected purely randomly and it was unclear what behavior, for example, a randomly selected block size has on the results. The M5 model again performs very well: The RMSE is 3.56 ms for the read model and 2.44 ms for the write model. The MAPE is 9.27% and 10.39% respectively. When compared to the previous section these results are worse because of the random gathering of the results.

When looking at the linear regression with all interaction terms the MAPE is 13.87% and 10.01% for the read and the write models. The RMSE is 4.81 ms and 2.60 ms respectively. These are very good values which are only slightly worse than those of the synthetic test in the section before.

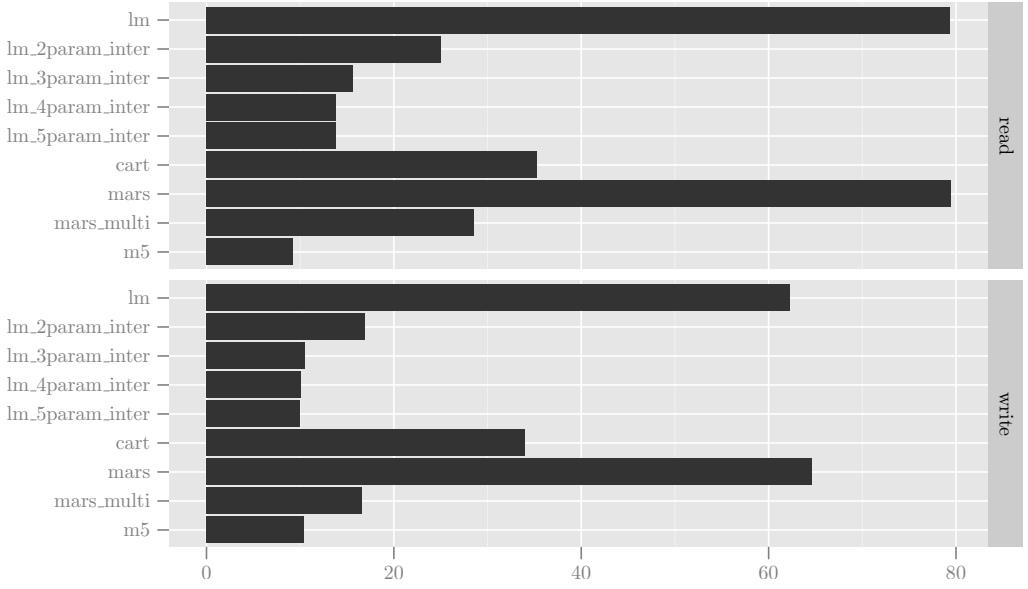
When regarding the multidimensional MARS model, the values are 6.49 ms and 2.84 ms for the RMSE and 28.52% and 16.64% for the MAPE. Like in the section before, these values are too high to be regarded a good fit.

The experiment conducted here shows more expressiveness than the one in the section before. Both sections show that the linear regression model containing all interaction terms and the M5 model have an acceptably low error for both, read and write models. One can conclude that the interpolation ability of the models is good. As the interpolation is one of the most important features, this first major result of the thesis is very promising.

In the next section, the discussion is extended to those values which are out of the range of the parameters. This means that the extrapolation abilities of the models are checked.



(a) RMSE in ms



(b) MAPE in %

Figure 7.2.: Comparison of the interpolation abilities of the nine regression models tested using 200 newly collected randomly selected samples.

Model	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
lm	11.32	100.32	8.73	74.67
lm_2param_inter	5.55	53.68	4.86	25.97
lm_3param_inter	3.76	28.09	4.81	23.05
lm_4param_inter	3.68	23.74	4.80	21.82
lm_5param_inter	3.67	23.22	4.76	21.32
cart	7.42	57.51	7.61	53.18
mars	11.40	101.62	9.38	81.81
mars_multi	4.62	44.86	5.23	29.14
m5	2.96	15.67	4.91	20.99

Table 7.3.: Comparison of the extrapolation abilities when reducing the original sample set by removing the extreme values and reuse the removed samples for training. This is the underlying data for the bar plots in Figure 7.3.

7.1.3. How good do the regression models extrapolate when using synthetic test sets?

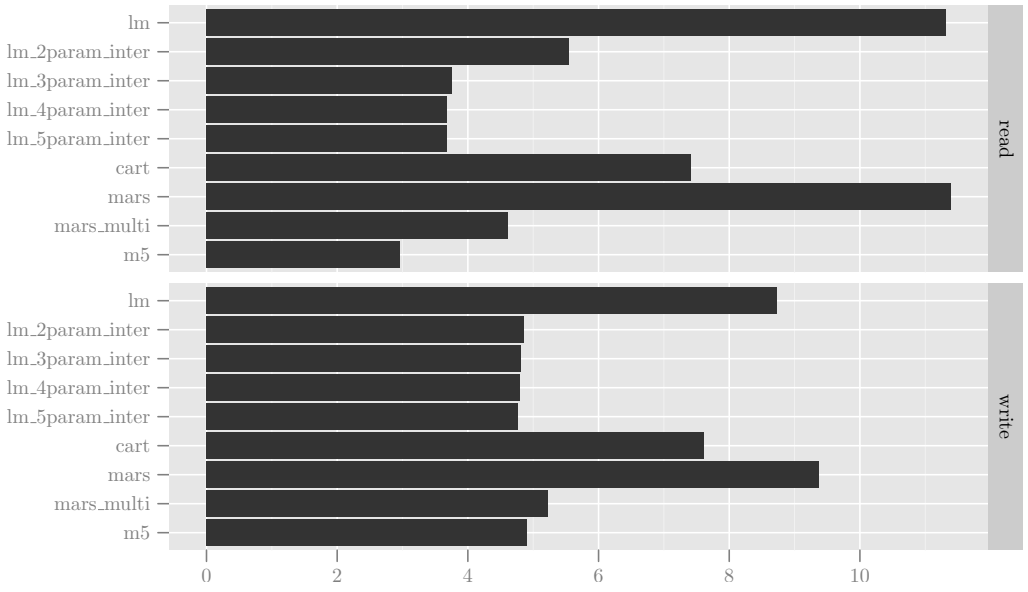
This question discusses the quality of the predictions out of the modeled range. This plays an important role, because not all possible configurations can be benchmarked beforehand. Again, the metrics for this question are RMSE and MAPE. Because some parameters have a closed range, extrapolation does not make sense for those parameters: The read percentage is already benchmarked from 0 to 100% and obviously no predictions out of this range can be made. The sequential/random access flag cannot be extended to more values. The prediction of values for other schedulers out of the existing data is not feasible. For the five parameters which are analyzed in this thesis the file set size and the block size are the only two parameters where extrapolation is an issue.

To test the extrapolation abilities of the models, this section takes a synthetic approach which solely relies on the already collected samples. In contrast to this, the next section judges on the extrapolation using newly collected samples. The advantage of this approach is that no new samples have to be collected. As the benchmarking of new configurations is time consuming, judging on the extrapolation without gathering new results can be a desirable option.

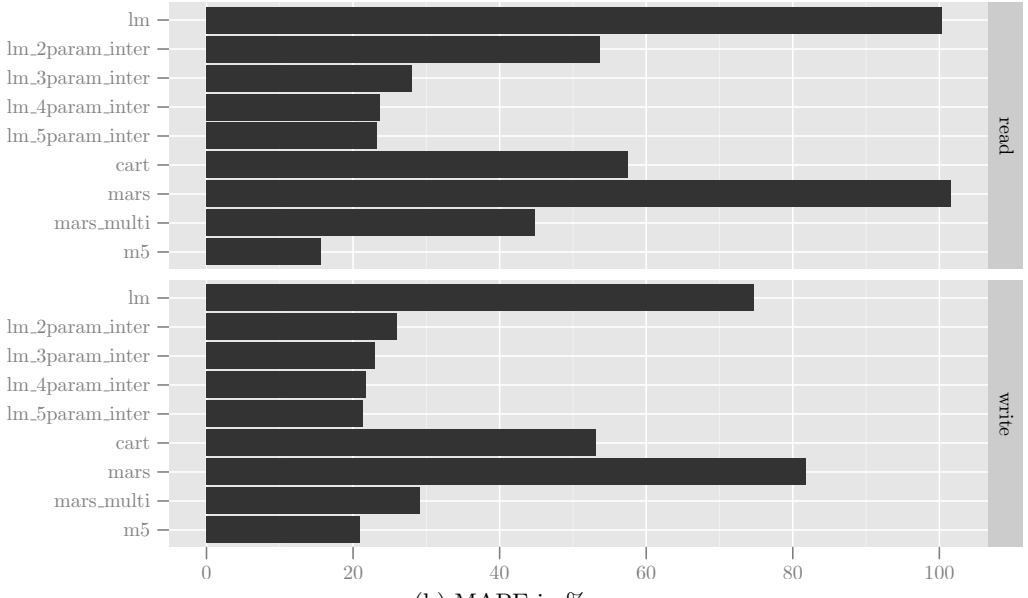
All configurations which meet one of the following four conditions are removed from the sample set:

- The block size is 4 kB.
- The block size is 32 kB.
- The file set size is 1 GB.
- The file set size is 100 GB.

This procedure removes the extreme values for both, block size and file set size from the sample set. 616 of the 1120 samples are removed. This leads to a remaining sample set of 504 elements. The reduced data set is then used to train the nine regression models defined above. The regression models which were built using the reduced data set were then tested. The test data consists of 100 randomly selected samples which were drawn out of the 616 samples which were removed in the initial step. This approach reduces the training set to get the possibility to test using the extreme values and therefore judge on the extrapolation abilities of the models. The disadvantage of this approach is that only 504 samples were available for the model creation. Especially for regression techniques which need a lot of data to fit the model, the results are not expected to be very good.



(a) RMSE in ms



(b) MAPE in %

Figure 7.3.: Extrapolation abilities of the regression models, trained using a reduced data set and tested using the left out samples.

The four bar charts in Figure 7.3 and the Table 7.3 show the results of this test. The overall results from this approach are very good considering the conditions under which the models have been generated.

The M5 model again performs the best: The MAPE is 15.67% for the read model and 20.99% for the write model. The RMSE is 2.96 ms and 4.91 ms respectively. In contrast to the previous section, this time the read model performs better than the write model. The write model suffers more from the reduction of the training set.

For the linear model containing all interactions possible (`lm_5param_inter`), the RMSE is 3.67 ms for the read model and 4.76 ms for the write model. The MAPE is 23.22% and 21.32% respectively. Again the write model does not perform better than the read model as observed in the section before. The advantage of M5 compared with the linear regression model gets smaller if the training set is reduced. For the write models, the RMSE of the linear regression model is even lower than the RMSE of the M5 model.

One can expect a better extrapolation if the model is built using the full sample set and additional values which are out of the range of this original sample set are used to test the model. This approach is taken in the next question.

7.1.4. How is the extrapolation ability of the regression models when testing using newly collected data?

This section focuses on another approach on how to measure the extrapolation abilities of the regression models. In contrast to the question before, where the existing data is used to judge on the extrapolation abilities, this section focuses on the more natural approach: New measurements are obtained and used to test the model. As in the previous sections, the metrics are RMSE and MAPE. The metrics are calculated based on the newly generated test samples.

This approach leads to the fact that the whole 1120 samples which are described in Section 5.1 are used to train the nine regression models. To test the extrapolation, newly collected benchmark results are used: The new experiments are randomly generated using the following schema:

- *Block size*: The block size is randomly chosen out of the following set: 1 kB, 2 kB, 36 kB, 40 kB, 48 kB and 64 kB. These values are chosen because they are not included in the original block size range which has its minimum at 4 kB and its maximum at 32 kB.
- *File set size*: The file set size is also randomly sampled using the following set: 256 MB, 512 MB, 110 GB and 130 GB. The original range of the file set size is 1 GB to 100 GB. Values larger than 130 GB can not be benchmarked due to size constraints: The benchmarked storage volume cannot hold more than 130 GB of data.
- *Scheduler*: Random selection of either CFQ or NOOP scheduler.
- *Read percentage*: The same value as in the original experiments: From 0% to 100% in seven steps.
- *Sequential access*: Both, random and sequential access were benchmarked.

This schema uses the original values for all parameters except for the block size and the file set size. Out of these $6 \cdot 4 \cdot 2 \cdot 7 \cdot 2 = 672$ possible configurations, 200 configurations are randomly selected as test set. This test set is benchmarked running for five times

Model	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
lm	17.59	97.27	15.83	107.14
lm_2param_inter	10.67	53.36	13.62	77.82
lm_3param_inter	9.38	32.83	14.89	55.67
lm_4param_inter	10.60	20.52	15.42	42.12
lm_5param_inter	10.57	20.16	15.41	41.93
cart	14.41	61.29	12.50	74.89
mars	17.41	95.92	16.08	110.33
mars_multi	8.65	41.01	12.90	64.59
m5	5.23	12.58	13.72	45.51

Table 7.4.: Comparison of the extrapolation abilities when using the original models and testing using random newly benchmarked data. This data is depicted as bar plots in Figure 7.4.

one minute. The results from these five runs are averaged like the original values in the previous sections.

All 200 test samples are predicted using the nine models. Then the predicted and the actual response time are compared. Using these two values, the RMSE and MAPE for each of the models are calculated. Figure 7.4 and Table 7.4 contain the values for each of the models.

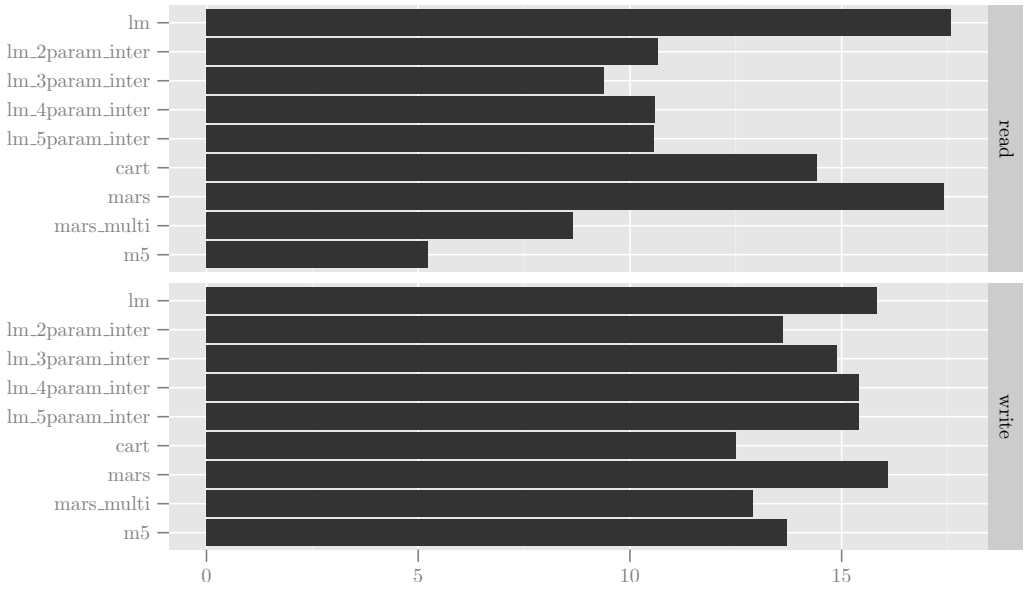
For this analysis the focus lies on the linear regression model with all interaction terms and on the M5 model. These models are focused because they have already proven to be the best performing out of the nine models benchmarked. Additionally they also perform best in terms of the extrapolation in this question and the question before.

For the read model, the results are still in an acceptable range when looking at the MAPE: It is 20.16% for the linear regression model and 12.58% for the M5 model. The results are even slightly better than those of the previous section. In contrast to this, the RMSE is very high for the linear model: The values have nearly tripled comparing to the previous section. The M5 model does not show such a heavy increase in the RMSE. Nevertheless, the value is also almost doubled.

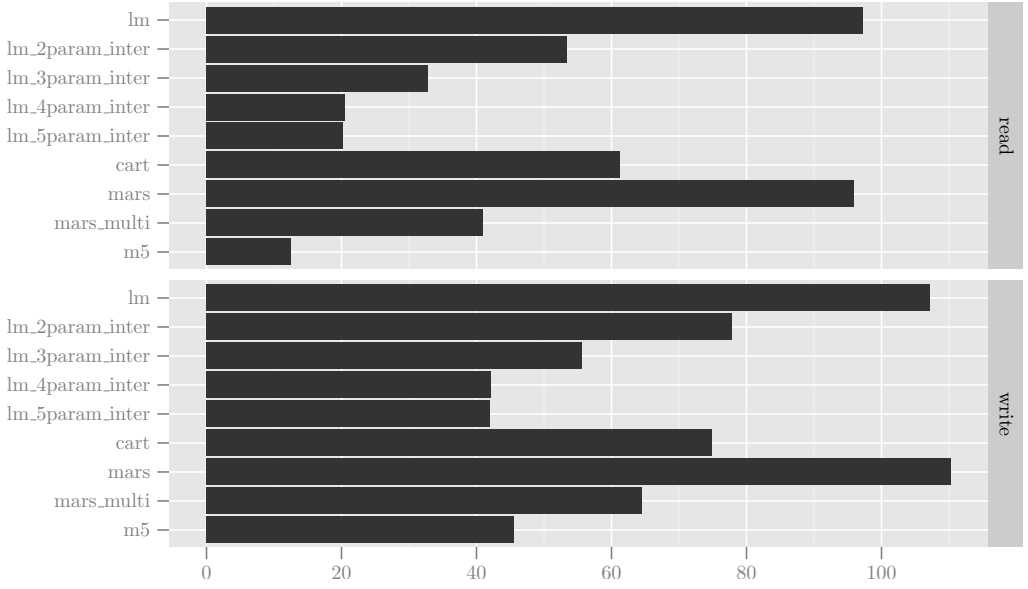
For the write model, the results are not promising: The MAPE is over 40% for both, the linear model and the M5 model. The RMSE is over 12 ms. These bad extrapolation abilities and the huge difference between the read and the write model results from the fact that the configurations predicted are not within the modeled range. These parameters show a different behavior which could not be included in the models because the training data did not cover these areas. When comparing the MAPE values, the linear regression model even performs better than the M5 model for write requests.

Such a big increase in the RMSE can only result from some samples which were predicted with a very big error. By inspecting the prediction results in detail it can be shown that most of the RMSE results from the upper block sizes. The three block sizes 40 kB, 48 kB and 64 kB show very bad predictions. The reason for this behavior is depicted in Figure 7.5.

The test set is split into two sets using the block size as condition: A lower block size set is constructed containing the two block sizes smaller or equal to 2 kB. The upper block size test set contains the block sizes greater or equal to 36 kB. When looking at the RMSE values for the read model the reason for the high overall RMSE values for the combined set becomes clear: As explained above the higher block sizes have a heavily increased RMSE compared to the lower block sizes. The reason for this behavior lies in the system limits:



(a) RMSE in ms



(b) MAPE in %

Figure 7.4.: Extrapolation abilities of all models, tested using randomly newly benchmarked data.

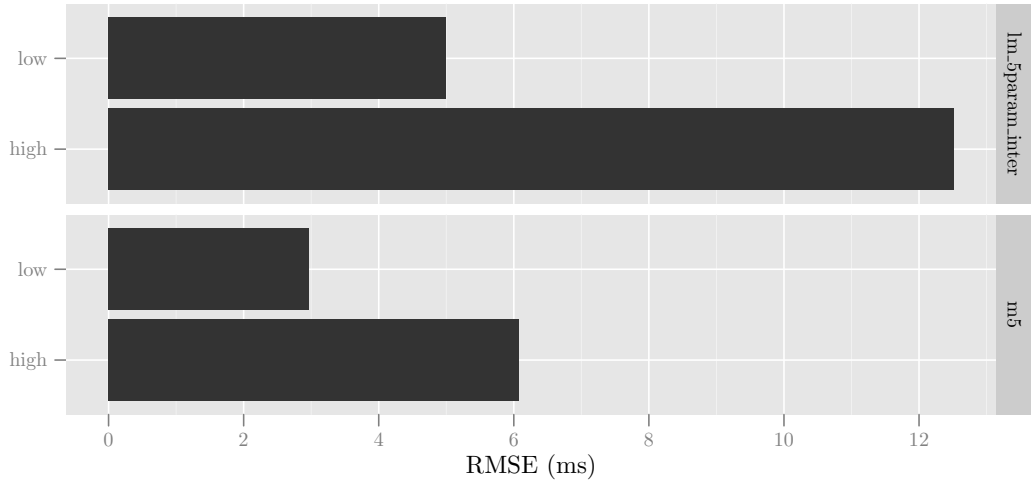


Figure 7.5.: Extrapolation abilities for read requests of the linear model including all interactions (`lm_5param_inter`) and the M5 model (`m5`). The test data was split into two sets: a "low" set containing the block sizes $\leq$ 2 kB and the "high" set containing the block sizes $\geq$ 36 kB. The metric depicted is RMSE.

During the benchmarking it showed that the fiber channel link to the storage device is fully saturated when using block sizes bigger than 36 kB. This introduces a non linear relationship which is not included in the models because no data containing block sizes bigger than 36 kB is used for their training.

The results from this question show that the extrapolation abilities of the models are not good. The main reason for this is that the systems behavior in the upper block sizes was not included in the models. This makes extrapolation very hard for the models. The extrapolation would become better if the upper block sizes had be included in the models. This change also requires a regression technique which can model the non linearity, for example MARS.

7.1.5. How many measurements are needed for an accurate model?

As explained above, 1120 samples are used to generate the models from the previous sections. It takes a long time to gather these results. For this reason, the question if similar regression models could be constructed using less samples and less detailed measurements arises. To answer this question, the same metrics as for the previous questions are used.

To check if a smaller number of samples is sufficient for the model generation, the following two subsets of the original sample set are defined: The first reduced sample set contains only the two extreme values for each parameter. The second subset contains three values by adding the middle value between the two extremes. For those parameters which only have two values (sequential access and scheduler), only two values are included. A special case is needed for the read percentage: A value of 0% leads to no read request issued at all. For this reason taking only the two extremes 0% and 100% into consideration does not work. Therefore the second lowest and second highest values are chosen. For the second reduced sample set a third point in the middle is added.

The leads to the two configurations `red1` and `red2` shown in Table 7.5. Additionally the table contains the number of configurations which are included in the sample set. When comparing these numbers to the original count of 1120 samples both counts show a great reduction. The benchmark time would have been heavily reduced if only these samples had been collected.

	red1	red2
Block size	4 kB, 32 kB	4 kB, 16 kB, 32 kB
Read percentage	25%, 75%	25%, 50%, 75%
File set size	1 GB, 100 GB	1 GB, 50 GB, 100 GB
Access	random, sequential	random, sequential
Scheduler	NOOP, CFQ	NOOP, CFQ
# of configurations	32	108

Table 7.5.: Two reduced samples sets which are used to test if accurate models can be generated using a smaller number of samples.

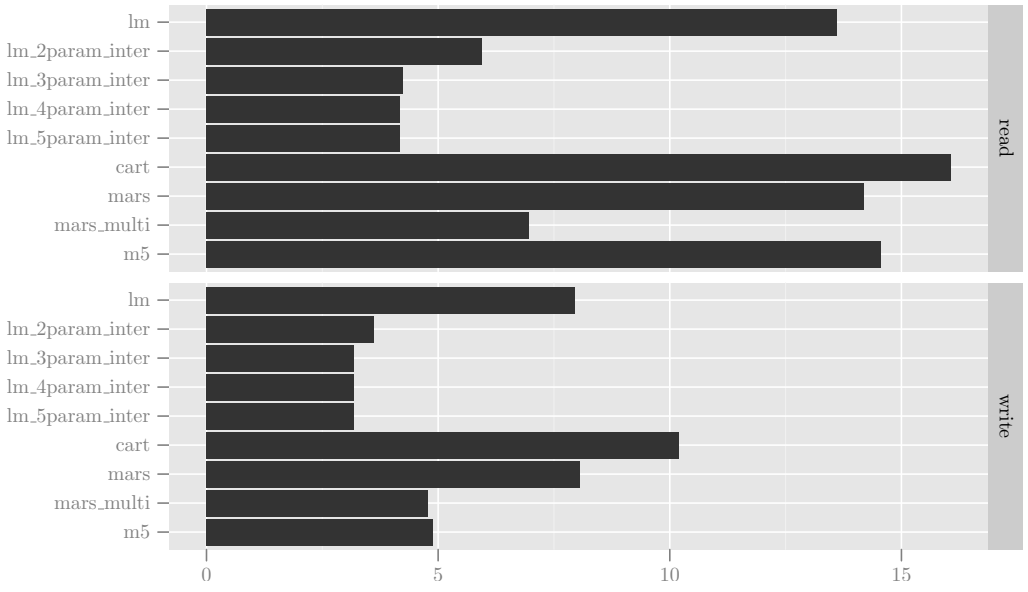
Using these two reduced test sets, four new sets of models are generated: One for every reduced test set and one for read and write requests. The models are tested using 200 random samples from the original 1120 sample set. Figure 7.6 and Figure 7.7 contain the results of this experiment as bar plots and the raw values as table. There are multiple things to notice when looking at the figures:

CART performs very badly: With a MAPE of 100% and an RMSE of over 15 ms the results get at lot worse compared to the model generated from the full data set. Although the results get better when a third point is included in the `red2` sample set the results remain bad: The MAPE is 43% and the RMSE is 7.8 ms. This behavior can be easily explained: The algorithm only splits a node if it represents enough samples. This leads to the fact that CART does not perform very well when operating in a situation where not enough data is available. When looking at the CART model itself this becomes even clearer: For the `red1` data set the CART model only contains three nodes with two leaves, for the `red2` model it contains 15 nodes with 6 leaves. Both are not large enough to model the complex behavior of the system.

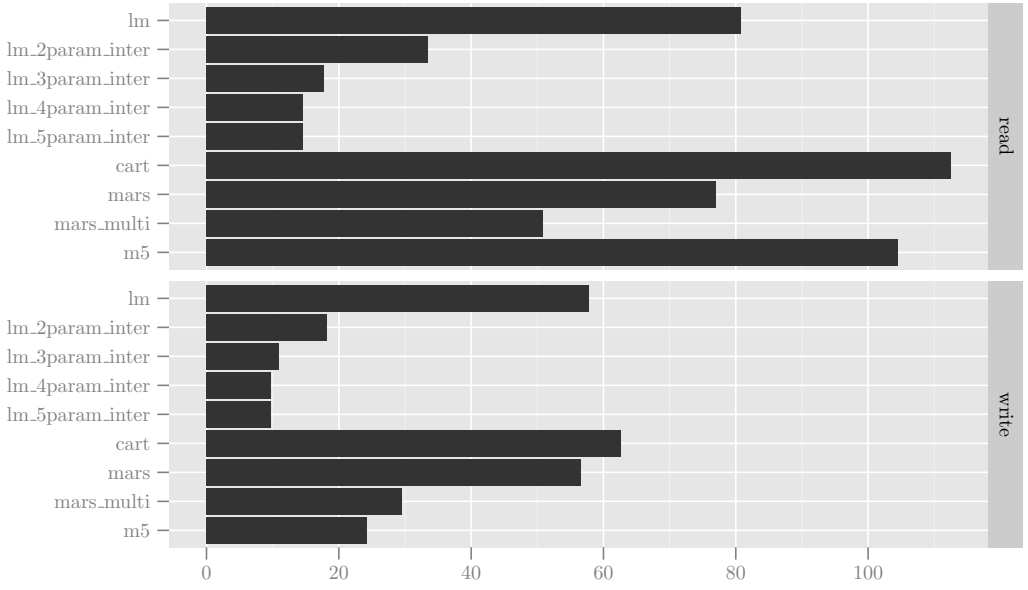
MARS also has its problems with the reduction of the training set. Especially with the smaller `red1` set the MAPE goes up to 48% for read and 28% for write requests. When trained using the original data in the previous section, these values are 36% and 17%. In contrast to this, MARS performs quite well when using the slightly bigger `red2` test set. With a MAPE of 36% and 19% the quality of the models are nearly the same as when trained using the full data set.

The M5 model, which has proven to be the best in all previous situations also has its problems. When looking at the smaller training set `red1`, the M5 model also has a MAPE of over 100% and an RMSE of 14.5 ms. Although the M5 algorithm was explicitly designed for cases with many parameters and a low number of training samples, the 32 samples are not enough for the M5 algorithm to generate an acceptable model. When looking at the second training set `red2`, the situation has changed: The inclusion of additional 76 samples in the training set makes the M5 model perform much better: The MAPE goes down to 16.7% for the read model and 14.7% for the write model. The RMSE is also heavily reduced. From these observation it become clear that M5 needs at least some data to produce good results.

The linear regression model performs quite well. In fact the linear regression models with many interaction terms do not suffer much from the reduction of their training data. The `lm_5param_inter` model has a MAPE of 14% for the read model and 10% for the write model when using the `red1` training set. The MAPE value for the original model which was trained using 35 times more data is only slightly better: 13% for the read model and 9% for the write model. This can be explained easily: The linear model cannot adapt to additional points. If the two extremes are measured well enough, then the linear regression can find a very good model. This is only true if there is a linear relationship in the data. In the previous section it was shown that there is in fact a linear relationship in the



(a) RMSE in ms

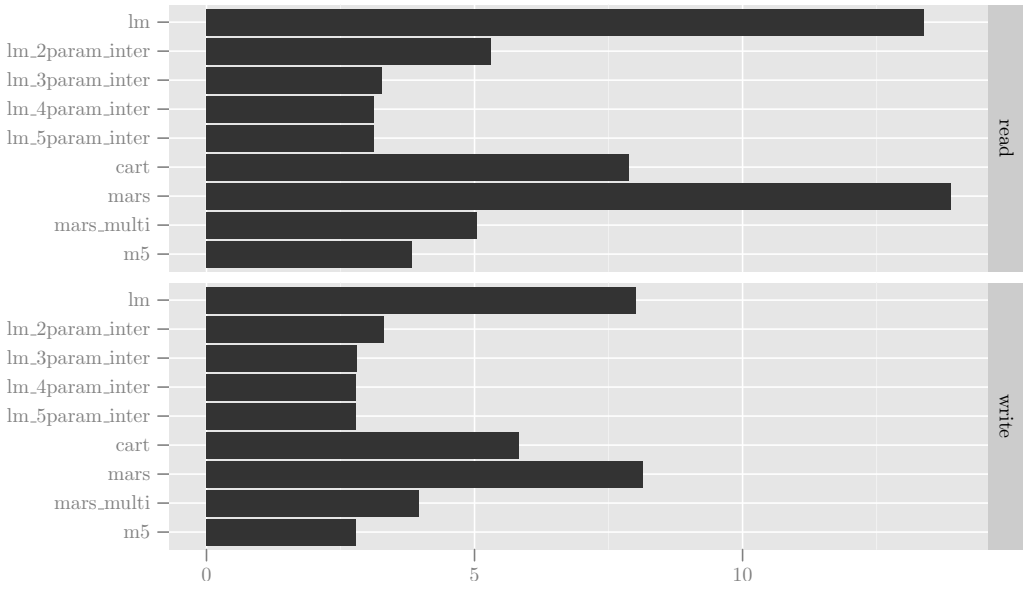


(b) MAPE in %

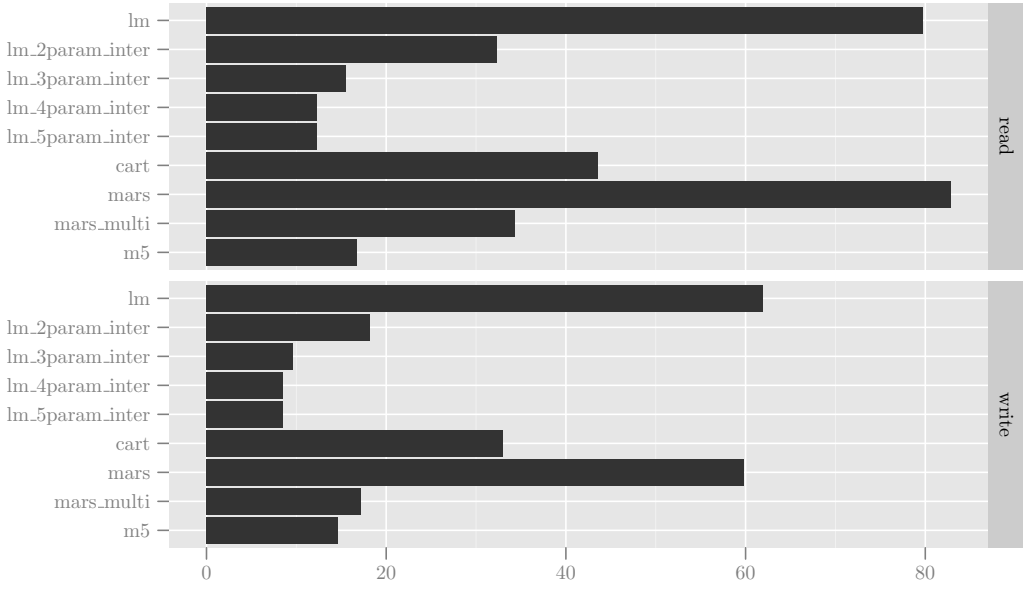
Model	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
lm	13.61	80.76	7.95	57.87
lm_2param_inter	5.96	33.42	3.61	18.22
lm_3param_inter	4.24	17.78	3.19	10.95
lm_4param_inter	4.18	14.63	3.19	9.78
lm_5param_inter	4.18	14.64	3.19	9.78
cart	16.08	112.57	10.20	62.62
mars	14.19	76.99	8.07	56.67
mars_multi	6.96	50.92	4.77	29.48
m5	14.56	104.46	4.88	24.24

(c) Data

Figure 7.6.: Comparison of the quality of the nine regression models built using the reduced data set `red1`. The models were tested using 200 randomly drawn samples from the complete sample set.



(a) RMSE in ms



(b) MAPE in %

Model	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
lm	13.39	79.79	8.00	61.95
lm_2param_inter	5.31	32.37	3.31	18.19
lm_3param_inter	3.27	15.50	2.81	9.68
lm_4param_inter	3.13	12.30	2.78	8.53
lm_5param_inter	3.13	12.29	2.79	8.51
cart	7.89	43.61	5.83	32.96
mars	13.89	82.89	8.14	59.87
mars_multi	5.05	34.35	3.97	17.15
m5	3.82	16.70	2.80	14.66

(c) Data

Figure 7.7.: Comparison of the quality of the nine regression models built using the reduced data set `red2`. The metrics were calculated using 200 randomly drawn samples from the whole 1120 sample set.

benchmarked ranges.

To summarize this answer: If there is a pure linear relationship, the reduction of the training samples does not produce a much bigger test error. This is only true for those models which are based on linear models, namely linear regression using least square and MARS. Other regression techniques like CART do not work well with a reduced data set, they need as much data as possible. Although based on linear models, M5 needs a minimum amount of samples to calculate a good model.

7.1.6. How can the regression modeling of nominal scale parameters be improved?

This question discusses the effects of the nominal scale parameters on the quality of the models. The metrics which are used are again MAPE and RMSE.

As explained at the beginning of this chapter, some parameters are not on an interval scale but on a nominal scale. Two out of the five parameters evaluated in this thesis are on a nominal scale: The I/O scheduler with its two benchmarked values NOOP and CFQ. This is not a fixed set: There are other schedulers which could be included in the benchmarking process and therefore in the regression models. The other parameter on a nominal scale is the sequential access flag. It can only take the values 1/True and 0/False.

As explained previously, all regression techniques examined in this thesis except CART do not support nominal scale parameters. For this reason the nominal scale parameters are transformed to an interval scales by assigning fixed values. This is only a work–around for the inability of the models. For this reason this section contains a new approach to these parameters:

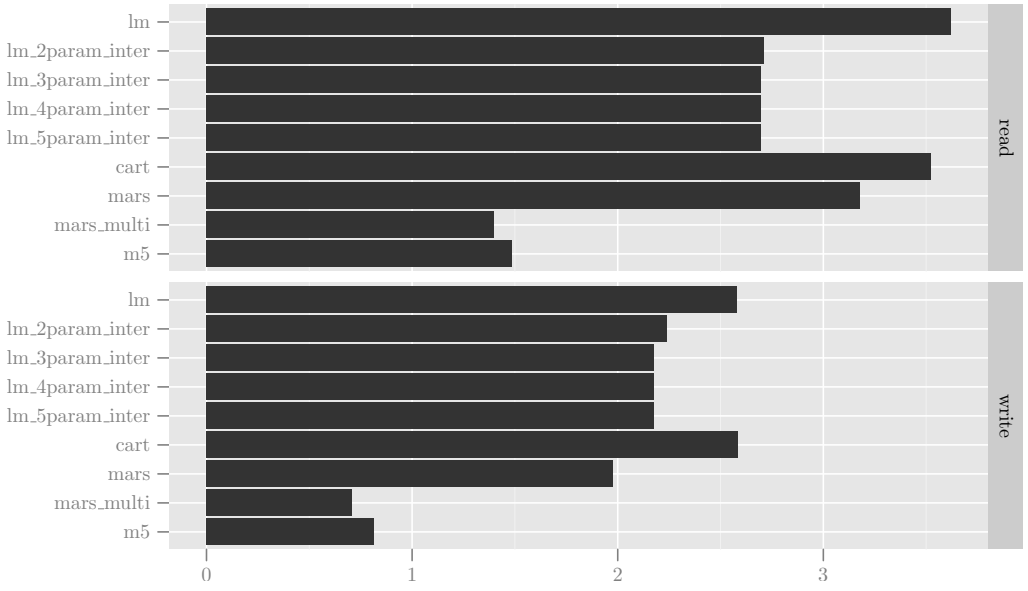
The samples collected are divided according to the values of the nominal scale parameters: Four new sample sets are created:

- Sequential access and NOOP scheduler
- Random access and NOOP scheduler
- Sequential access and CFQ scheduler
- Random access and CFQ scheduler

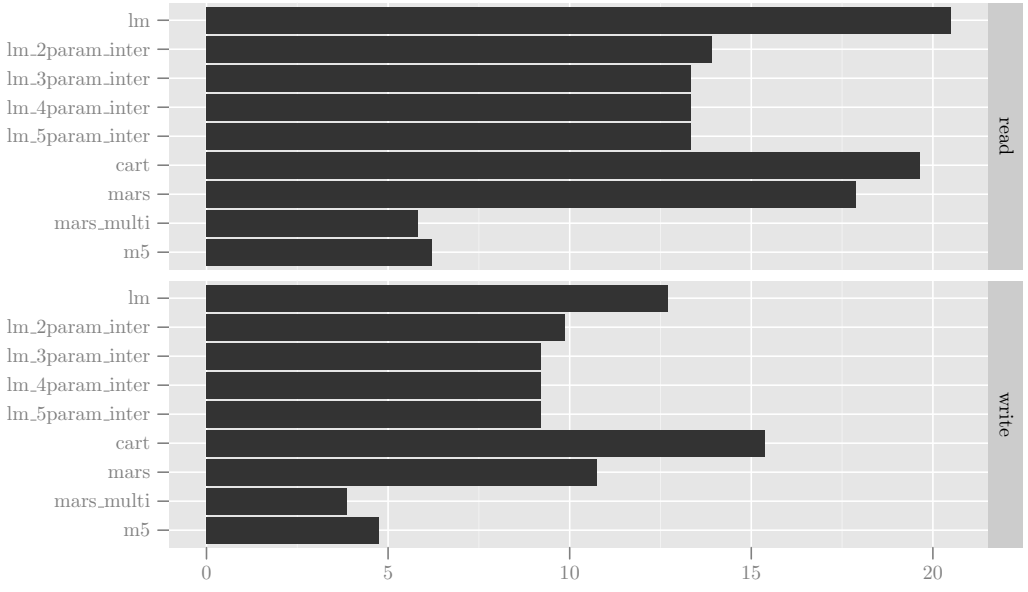
The four sample sets are of equal size. Each of the four sample sets is used to train and test the nine regression models which are used throughout this thesis. The validation is done by using 10-fold cross–validation. This process leads to four times nine models: Nine models for each of the four sample sets specified above. For each of the regression techniques and each of the four data sets, the MAPE and the RMSE are collected. To compute the MAPE and RMSE of the complete and original data set which is used in the other sections, the MAPE and RMSE values from the four models are averaged for each of the regression models.

Each of the regression models contains three remaining parameters: As the scheduler and the sequential access flag have constant value in each of the four sets, the remaining parameters are the file set size, the block size and the read percentage. All these remaining parameters are on an interval scale and are therefore better suited for the regression techniques.

If additional schedulers were benchmarked or more nominal parameters were examined, the number of models would increase linearly: Benchmarking another scheduler for example increases the number of models to six.



(a) RMSE in ms

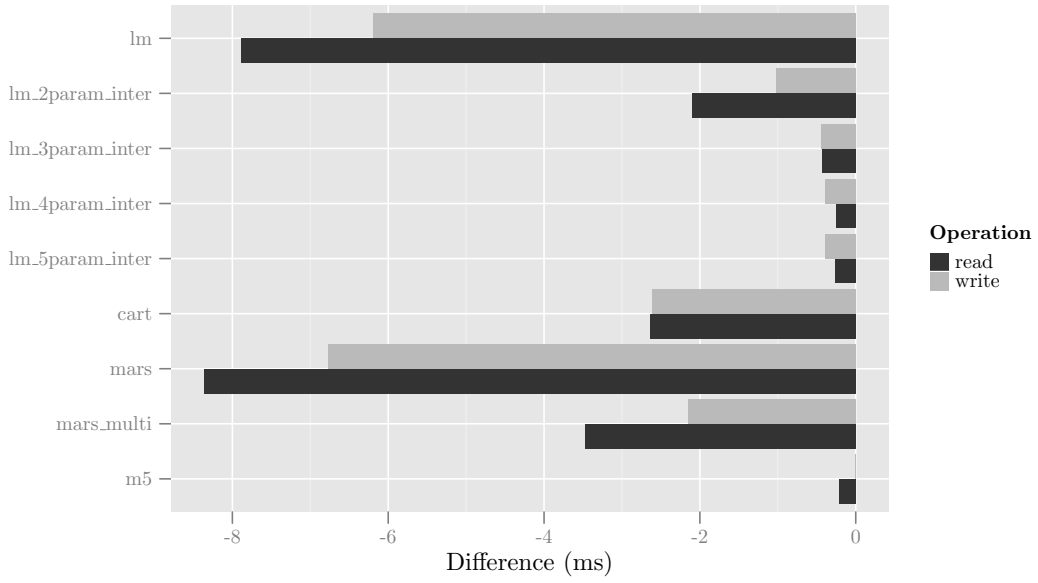


(b) MAPE in %

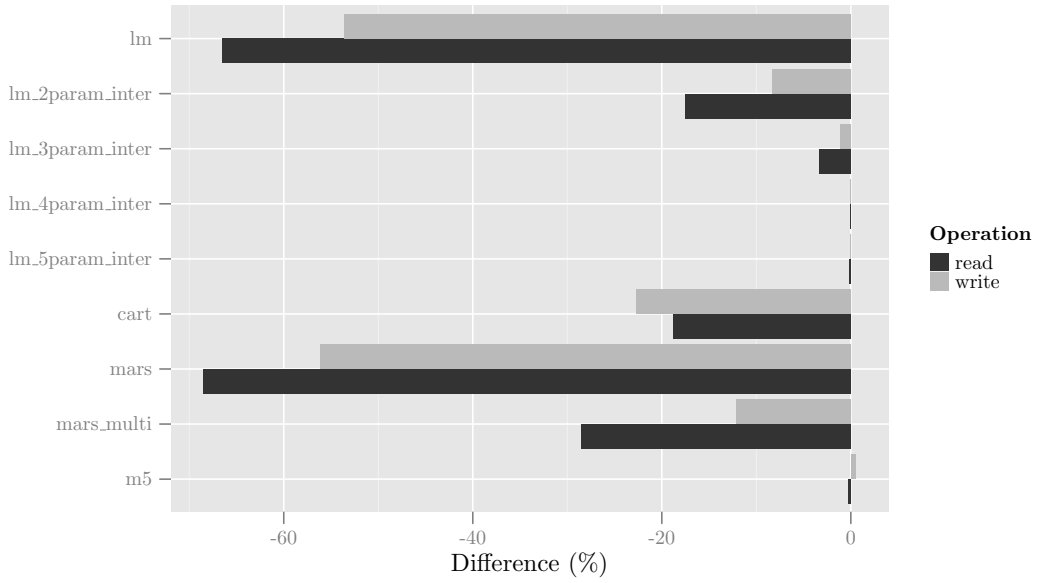
Figure 7.8.: Computed MAPE and RMSE values if separated models are built for each value of nominal scale parameters. This means that four models were created and the average of the RMSE and MAPE is depicted here.

Model	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
lm	3.62	20.51	2.58	12.72
lm_2param_inter	2.71	13.91	2.24	9.86
lm_3param_inter	2.70	13.35	2.18	9.20
lm_4param_inter	2.70	13.35	2.18	9.20
lm_5param_inter	2.70	13.35	2.18	9.20
cart	3.52	19.63	2.58	15.36
mars	3.18	17.89	1.98	10.76
mars_multi	1.40	5.82	0.71	3.86
m5	1.48	6.21	0.81	4.75

Table 7.6.: Generalizationa abilities of regression models which were built separate for the values of the nominal scales. Figure 7.8 contains a graphical representation.



(a) RMSE



(b) MAPE

Figure 7.9.: The split models, where four different models are used, compared to the original models using a single model. A negative value means an improvement due to the splitting, a positive value means decline due to the splitting.

The results of this approach can be found in Figure 7.8 and in Table 7.6. The linear models with interactions between more than two variables do not differ at all. The reason for this is, that there are only three parameters included in the models which means that the `lm_3param_inter`, the `lm_4param_inter` and the `lm_5param_inter` models are identical.

In Section 7.1.1, the same data was used to build one model whereas in this approach four models were generated. Figure 7.9 contains a comparison between the new models generated in this section and the original simple models without the splitting. When comparing the results, it becomes clear that all models benefit from the approach taken here: Those models which do not contain interaction terms show the hugest improvements: For the linear regression model without interactions the MAPE goes down from 87.02% to 20.51% for the read model. For the RMSE, the decrease is from 11.51 ms to 3.62 ms. The same improvements can be found in the data for CART: The MAPE goes down from 38.40% to 19.63%. The behavior of the MARS models shows an interesting change: The MAPE of the MARS read model without interactions improves from 86.36% to 17.89%. The multidimensional MARS model has a MAPE of 5.82% for the read model and 3.86% for the write model. The RMSE is 1.40 ms for the read model and 0.71 ms for the write model. These values are better than the more sophisticated M5 algorithm. The M5 models do not improve much: The MAPE goes down from 6.49% to 6.21%. Due to the huge change in the multidimensional MARS model, it performs best out of the nine models compared. Similar to the M5 model, the linear regression models with higher levels of interactions also do not show a huge improvement: For the `lm_5param_inter`, the MAPE goes down fro 13.47% to 13.35% for the read model.

The numbers show, that those models which did not perform very well using only a single data set improve a lot when using split data sets for the nominal scale parameters. Non of the models suffered from this change.

While this approach leads to a better performance of some algorithms it introduces another level of complexity. For some applications of the models it might be better to have a single model for all parameters, while for other uses, it might be acceptable to have four or more models.

A further approach which is possible but not evaluated here, is to combine different regression techniques: If, for example, one regression technique works better for sequential access and another works better for random access, these different models could be combined.

7.1.7. Summary

This section gives an overview of the answer to five previous research questions. To summarize the experiments which have been conducted in the previous sections, Table 7.7 contains a row for each section. It is specified which samples have been used for training and which have been used for testing. Additionally the results (MAPE and RMSE) of both, the read and the write models are noted. The metrics included in the table are those of the best performing model in the question. The term "Original sample set" references the set containing 1120 samples as explained in the introduction of this chapter.

7.2. Evaluation and Analysis of Regression Techniques

7.2.1. How does the generalization ability of the different regression techniques compare?

Nine different models using four different modeling techniques are evaluated in this thesis. This section discusses how these models compare in terms of the quality of fit and why the models show this behavior.

	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
Interpolation (synthetic) (Section 7.1.1) Trained & Tested using: 10-fold cross–validation using the original sample set Best model: M5 model	1.69	6.49	0.81	4.24
Interpolation (Section 7.1.2) Trained using: Original sample set Tested using: 200 randomly generated samples from the parameter ranges Best model: M5 model	3.56	9.27	2.44	10.39
Extrapolation (synthetic) (Section 7.1.3) Trained using: Original sample set with extreme values of each dimension taken away (504 samples) Tested using: Random samples drawn from the extreme values taken away in the training Best model: M5 model	2.96	15.67	4.91	20.99
Extrapolation (Section 7.1.4) Trained using: Original sample set Tested using: 200 randomly selected samples from outside the parameter ranges Best model: M5 model	5.23	12.58	13.72	45.51
Reduced Sample Set (Section 7.1.5) Trained using: Reduced sample sets (32 samples) Tested using: 100 randomly selected samples from original sample set Best model: Linear regression model including all interactions (`lm_4param_inter`)	4.18	14.63	3.19	9.78
Trained using: Reduced sample set (108 samples) Tested using: 100 randomly selected samples from original sample set Best model: Linear regression model including all interactions (`lm_5param_inter`)	3.13	12.29	2.79	8.51
Separate models for nominal scale parameters (Section 7.1.6) Trained & Tested using: 10-fold cross–validation using the original sample set divided into four sets using the values of the nominal scale parameters. Best model: Multidimensional MARS model (`mars_multi`)	1.40	5.82	0.71	3.86

Table 7.7.: Comparison of the different experiments conducted in the previous sections.

To compare the quality of different models, cross–validation has to be used. In Section 7.1.3 the interpolation strength is evaluated using cross–validation. The same results can be reused for this question. Looking again at Figure 7.1, which compares the RMSE and MAPE of different models using 10-fold cross–validation, these models can be roughly grouped into the following three relative categories:

- *Worst quality*: The simple, one degree MARS model and the simple linear model without interactions.
- *Average quality*: The multidimensional MARS model, the CART tree and the linear regression model containing interactions between two parameters.
- *Best quality*: The three linear regression models containing interactions between three and more parameters and the M5 model.

The first observation is that the models without interactions perform badly. This is true for both, the simple linear model without interactions and the MARS model which can introduce hinges in the function. There is a simple reason for this behavior: The storage performance benchmark results heavily dependent on interactions between multiple parameters. By modeling only the parameters and not their interactions no good model can be obtained. This is already shown in Section 6.2. An ANOVA is used there to show which parameters have an influence on the response time. It is shown that the single parameters are not sufficient to explain the response time. This should be kept in mind for the further discussion of the regression models. Nevertheless, the linear model performs slightly better than the MARS model. The main reason for this is the cost–factor in the MARS algorithm: In order to prevent overcomplicated models, the MARS algorithm penalizes each term and every hinge using a cost function. This means that while the linear regression model can compute a coefficient for every parameter, the MARS algorithm prevents this if the introduction of this coefficient does not improve the model enough.

As the multidimensional MARS, the CART tree, and all other linear regression models except from the simplest one can model higher degree interactions. Thus they perform better than those models without interactions. As shown by Hastie et al. [HTF11], CART can be seen as a special case of MARS. It is shown that the MARS algorithm can be used to construct CART models if further restrictions are applied. This explains why the multidimensional MARS performs slightly better than CART. Another explanation is the nature of the models: CART cannot model additive structures but instead must rely on a chain of conditions. Furthermore CART is even unable to construct linear increasing functions. It can only model stepwise functions. This introduces a disadvantage for the CART model as the results of the storage benchmarks are expected to have a linear relationship which can only difficultly be modeled using stepwise functions. Although using a simpler algorithm, the linear regression model which includes interactions between two parameters performs about the same as CART and the multidimensional MARS. The main reason for this is that the data which was benchmarked and used for the training is based on a linear coherency.

The best performing models are on the one hand the linear models which include interactions between three, four and five parameters. The addition of the interactions between three parameters has improved the models a lot which leads to the conclusion that these interactions still play an important role. This is especially true for the read model where the difference between the two models is bigger than for the write model. The addition of the interactions between four parameter does only improve the models in a small amount. The improvement which results from the addition of the interaction term between all five parameters can hardly be measured. It is so small that the models must be regarded nearly identical. Nevertheless, there is a very small difference: The RMSE is 0.006 ms lower and the MAPE is 0.04% lower for the model including the interaction term between all five

parameters. The conclusion from this is that this interaction term does not provide any valuable information to the model. This can be supported by the fact that coefficient of this term is $-1.633 \cdot 10^{-11}$, which is very close to zero.

The M5 model shows an impressive result: It outperforms the second best model, the linear regression model including all interactions with less than two thirds of the RMSE and MAPE. The combination of the advantages of CART and the linear regression models leads to very good interpolation abilities. It should be kept in mind that the linear models which are attached to the leaves of the M5 tree do not include any interactions. This behavior is solely modeled using the regression tree of the M5 model.

An important question is, why the multidimensional MARS model is modeling the data so much weaker than the linear models. As the multidimensional MARS model is allowed to add all interaction terms, one could expect it to be at least as good as the linear models which have the same constraints. MARS should perform even better as it is not fixed to strict linear functions but can introduce partly linear functions. The answer to this question lies in the MARS algorithm: As mentioned above this algorithm introduces a cost for each term involved. This leads to a reduction of the model to as least terms as possible. This might even take away whole interaction terms because the algorithm regards them as too expensive. In contrast to that, the linear regression can introduce a coefficient for every parameter and every interaction term. No interaction term has to be left away due to costs. If a term is not needed, the coefficient can simply be zero. This explains why the linear models perform better than MARS. Additionally the data seems to be of very linear nature. For this reason, the big advantages of MARS, the partly linear functions, cannot show their strengths on this data set.

As discussed in Section 7.1.4, configurations other than those benchmarked do not show this linear behavior: If the block size is increased above the current limit of 32 kB, the assumption that there is a linear correlation does not hold anymore. This means that the linear model make a large profit from the current selection of the benchmarking ranges. If other ranges had been selected for the ranges, the MARS model could have performed better compared to the linear regression models. Additionally the linear behavior is fixed to the system benchmarked. One cannot assume that a similar linear correlation between the parameter can be made on other systems even with the same parameter ranges. A simple example is a system with only half the bandwidth for the storage device connection. On such a system the storage connection gets saturated quicker and with lower block sizes.

In general, it should be kept in mind that this comparison is only valid for the nine models explained above and only for this specific data set. Nevertheless, the approach specified here can be used on other systems and using other ranges, but the conclusions could be different.

7.2.2. What are the advantages and disadvantages of the modeling techniques?

This question discusses the advantages and disadvantages of the different modeling techniques apart of the quality which is discussed in the previous question. The metrics which are used here to facilitate the comparison are:

- Time needed to construct a new model.
- Time needed to predict values.
- Interpretability of the resulting models.
- Complexity of the algorithms and their implementation.

Model	Modeling		Prediction	
	Read (s)	Write (s)	Read (s)	Write (s)
lm	0.00500	0.00500	0.00167	0.00115
lm_2param_inter	0.00800	0.00800	0.00135	0.00135
lm_3param_inter	0.01200	0.01200	0.00167	0.00156
lm_4param_inter	0.01400	0.01500	0.00188	0.00167
lm_5param_inter	0.01500	0.01400	0.00177	0.00177
cart	0.02500	0.02500	0.00115	0.00104
mars	0.04400	0.04400	0.02083	0.02042
mars_multi	0.05800	0.05600	0.02125	0.02146
m5	0.14600	0.18100	0.05052	0.05156

Table 7.8.: Comparison of the time needed for the model creation ("Modeling") and for the prediction of 1000 samples ("Prediction"). The time needed for both, the read and the write models is depicted.

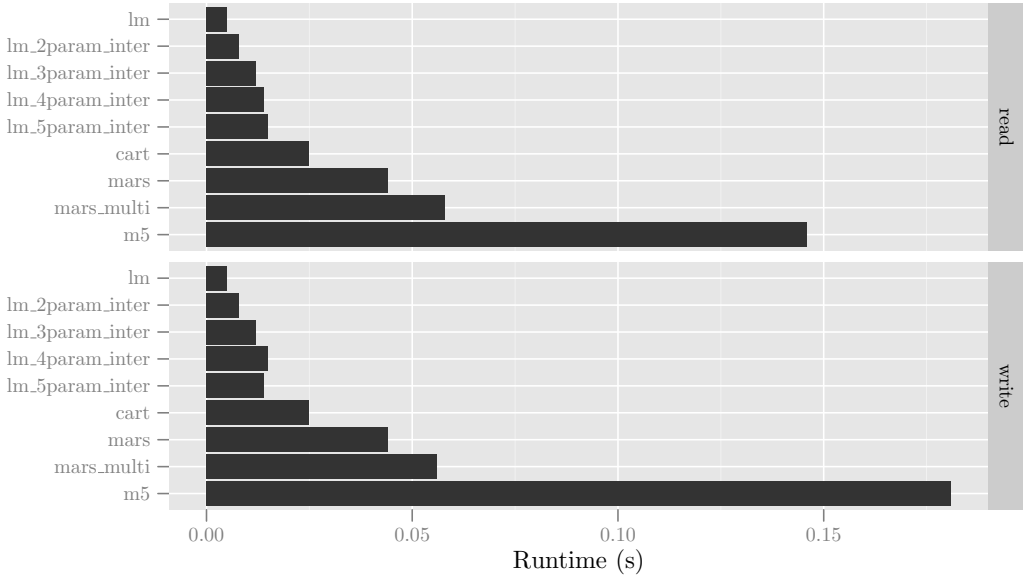


Figure 7.10.: Comparison of the time needed for the generation of a single model of each of the regression techniques.

While the first two metrics can be compared objectively, the last two metrics can only be interpreted subjectively.

The *time which is needed to construct a new model* is important if the models should be generated on the fly. This approach is taken if preexisting models get refined and adapted to current conditions while the system is running. Even for the approach taken in this thesis, where only an offline model generation is done, the generation time is important: Especially when comparing different modeling techniques, one cannot wait hours for the models to generate. It should be kept in mind that the time needed for model generation, and also the time needed for prediction, is dependent on the system configuration where the models are generated. In this case the analysis is executed on an Intel Core i5 M520 CPU with two cores running at 2.40 GHz. The execution time is clearly bound by the speed of the CPU and no other processes were running the model creation. The execution times are stable which means that repeated runs lead to nearly the same results. The execution time is also dependent on the algorithm implementation, in this case the default implementations shipped with R are used.

Figure 7.10 and Table 7.8 contain the time needed for constructing a new read and write model using the regression techniques explained above.

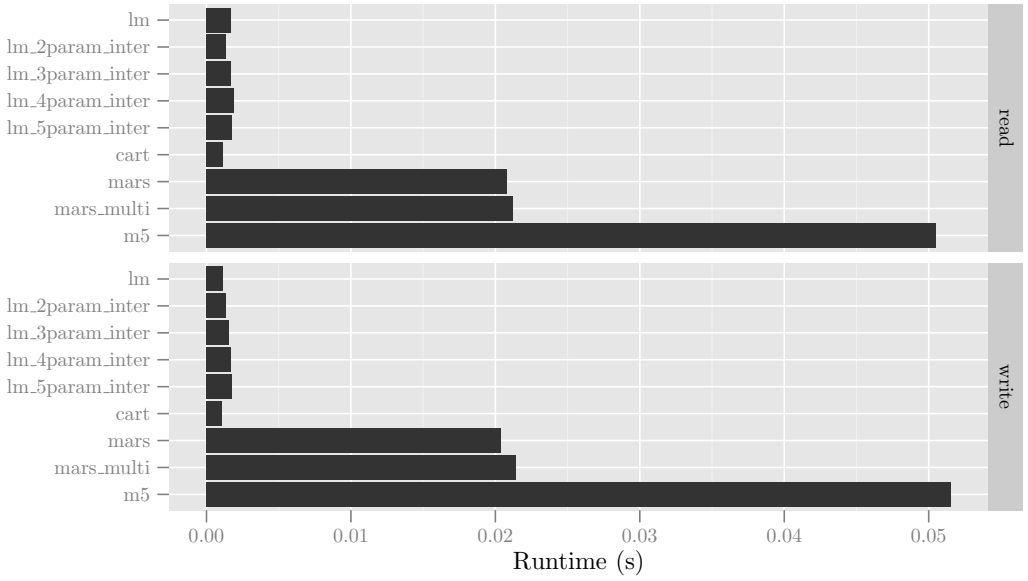


Figure 7.11.: Comparison of the time needed for the prediction of 1000 samples for each of the regression techniques.

There is no significant difference between creating a read and a write model. As both model contain the same amount of data any significant difference would be surprising. The figures clearly show that the linear regression models are the fastest of the technologies benchmarked. When comparing the `lm` and `mars` models, which both do not include interaction terms, the `mars` model is about six times faster. The difference also exists when comparing the `marsMulti` model with the `lm_5param_inter` model which both include every interaction term possible: The linear model is about 4 times faster than the multidimensional MARS. It becomes clear that the modeling of the hinges has its price. As these functionality has not proven to be of much value for the data modeled in this thesis, the run time is another factor which rule out the MARS models in this thesis case. With an absolute runtime of 0.005 s or about 5 ms the linear model can be computed easily at runtime. This can be supported by the fact that the current implementation is using a non–optimized statistical software. The computation of the least square regression technique can be done very efficiently because of its simple nature. The creation of the M5 model is nearly three times slower than the multidimensional mars model and nearly ten times slower than the comparable `lm_5param_inter` model. The generation of the best performing model has its price. The algorithm has to compute a linear regression model for every node created. As there are much more nodes created than node later contained in the results, this can explain the long run time. Additionally, the implementation of M5 is not optimized for performance.

The *prediction times* are depicted in Figure 7.11 and the underlying values can be found in Table 7.8. The prediction time plays an important role in online prediction but has an influence on every application using the models. If the predictions take to long to be calculated, the models become worthless. The figure shows for each of the models the duration of the prediction of 1000 samples. This number is chosen to get stable results, otherwise the runtime would be too small to measure accurately. The models can be categorized into two groups: The linear regression models and the CART model show a fast prediction time. The MARS and M5 models are much slower: The difference between the simplest MARS models (`mars`) and the most complicated linear regression model (`lm_5param_inter`) is huge: MARS is eleven times slower than the biggest linear model. The reason for this slowdown lies in the nature of the MARS models: While linear

	Read	Write
(Intercept)	21.95252	13.80373
schedulerNOOP	-18.17612	-16.80425
filesetSize	0.00014	0.00009
blockSize	0.00040	0.00045
readPercentage	-0.17071	-0.09890
sequentialAccess	7.31739	9.93321

Table 7.9.: Linear regression model without interactions. The table shows the values for the coefficients for both, the read and the write model. All values rounded to five digits.

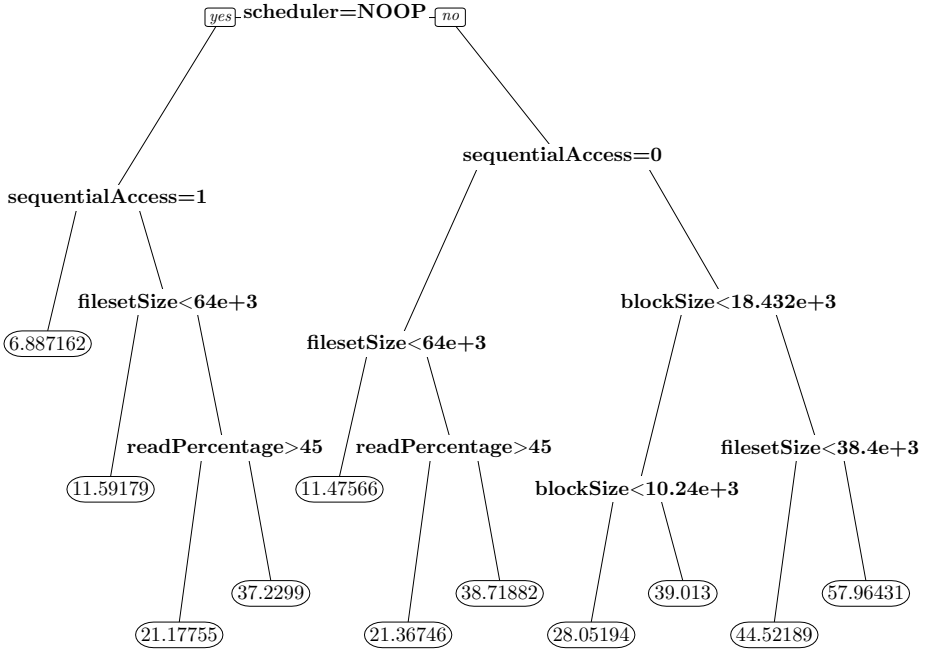
models involve simple multiplications and additions which can be done very effectively on CPUs, the MARS algorithm needs conditional statements and therefore branches to predict the values. Especially branches are very expensive on today's CPU architectures with their huge pipelines. Another reason for this bad prediction times is the implementation: The CART algorithm does not perform as bad as the MARS algorithm, although is also does include conditions. It seems that the implementation of MARS is not optimized for performance.

The M5 model is nearly another three times slower than the MARS model. It is also 28 times slower than the linear regression models. The complexity of the M5 model has its price when it comes to the evaluation of both, the regression tree and the linear models. As explained before, the implementation of M5 is explicitly not optimized for performance. Nevertheless, with a prediction time of 50 milliseconds for 1000 samples even the M5 model performs good enough.

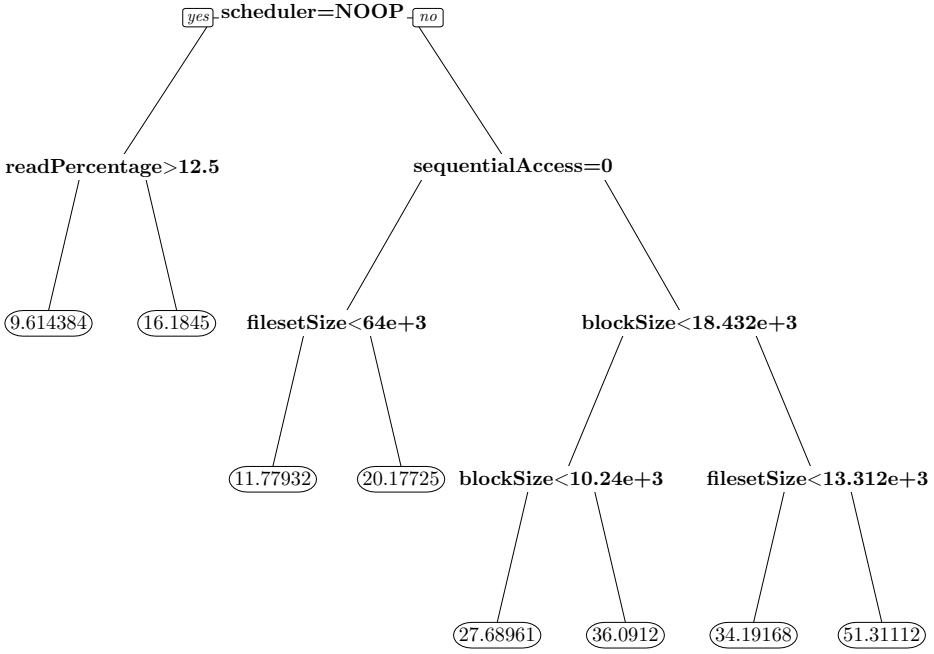
The next metric discussed here is interpretability. To compare the interpretability of the result, six different models are printed on the next pages:

The CART model (compare Figure 7.12) contains a decision tree. By looking at this tree it is easy to conclude the importance of the parameters: Important parameters are further up in the tree, towards its root. It is also easy to infer the influences of the parameters as the split points provide the information how these parameters influences the result. For example it becomes clear from the tree that there is some important change in the behavior of the model below a file set size of $64 \cdot 10^3 = 64$ GB. The next file set size benchmarked below 64 GB is 50 GB. This is exactly the cache size of the IBM DS8700 storage system. This shows that influencing values directly find their way to the CART model and can be seen easily. The CART model is also the only model which can be depicted as a figure. All other models cannot be plotted because of their huge dimensions. As it is often easier to see the coherences in a plotted diagram, rather than in a table full of numbers, this can be regarded a huge advantage of CART. Additionally, CART is the only algorithm which can easily adapt to factor levels. As explained above, the scheduler and the sequential access flag are nominal scale parameters. The CART model can simply include even parameters containing strings (like the scheduler) in its model. This further simplifies interpretation as no transformation is needed.

Table 7.9 shows the resulting linear regression models for a simple regression without interaction terms. Table 7.10 contains a similar table for the linear regression model with all interactions, including the interactions between all five terms. The three intermediate models are generated but are not printed here as they show similar results. The tables show the coefficients for each parameter plus an intercept term. Each of the coefficients is printed for both, the read and write model. The regression algorithm computes a coefficient for every parameter. Even terms with very small coefficients (near zero) are therefore included in the model. None of the coefficients is exactly zero but the absolute



(a) Read Requests



(b) Write Requests

Figure 7.12.: CART model for the whole data set.

	Read	Write
(Intercept)	5.74042	5.01551
schedulerNOOP	-0.68947	-0.04302
filesetSize	0.00059	0.00024
blockSize	0.00030	0.00049
readPercentage	-0.07190	-0.05869
sequentialAccess	23.81223	17.09872
schedulerNOOP:filesetSize	-0.00013	-0.00007
schedulerNOOP:blockSize	0.00008	-0.00002
schedulerNOOP:readPercentage	0.01017	-0.00366
schedulerNOOP:sequentialAccess	-26.53845	-20.02582
filesetSize:blockSize	-0.00000	-0.00000
filesetSize:readPercentage	-0.00000	-0.00000
filesetSize:sequentialAccess	-0.00067	-0.00023
blockSize:readPercentage	-0.00000	-0.00000
blockSize:sequentialAccess	0.00093	0.00048
readPercentage:sequentialAccess	-0.06271	-0.01441
schedulerNOOP:filesetSize:blockSize	0.00000	0.00000
schedulerNOOP:filesetSize:readPercentage	0.00000	0.00000
schedulerNOOP:filesetSize:sequentialAccess	0.00021	0.00006
schedulerNOOP:blockSize:readPercentage	-0.00000	0.00000
schedulerNOOP:blockSize:sequentialAccess	-0.00088	-0.00044
schedulerNOOP:readPercentage:sequentialAccess	0.12030	0.07434
filesetSize:blockSize:readPercentage	0.00000	0.00000
filesetSize:blockSize:sequentialAccess	0.00000	0.00000
filesetSize:readPercentage:sequentialAccess	0.00001	0.00000
blockSize:readPercentage:sequentialAccess	-0.00001	0.00000
schedulerNOOP:filesetSize:blockSize:readPercentage	-0.00000	-0.00000
schedulerNOOP:filesetSize:blockSize:sequentialAccess	-0.00000	-0.00000
schedulerNOOP:filesetSize:readPercentage:sequentialAccess	-0.00000	-0.00000
schedulerNOOP:blockSize:readPercentage:sequentialAccess	0.00001	-0.00000
filesetSize:blockSize:readPercentage:sequentialAccess	-0.00000	-0.00000
schedulerNOOP:filesetSize:blockSize:readPercentage:sequentialAccess	-0.00000	0.00000

Table 7.10.: Linear regression model with all interaction terms, including those between two, three, four, and five parameters. The coefficients for both, the read and write model, are shown in the columns. All values are rounded to five digits.

20.29673		
-18.17612	*	schedulerNOOP
+0.0001511071	*	h(filesetSize-25600)
-0.0001162011	*	h(25600-filesetSize)
+0.0003646261	*	h(blockSize-12288)
-0.0005100038	*	h(12288-blockSize)
-0.1084117	*	h(readPercentage-50)
+0.3249405	*	h(50-readPercentage)
+7.317385	*	sequentialAccess

(a) Read Requests

23.28173		
-16.80425	*	schedulerNOOP
+6.101831e-05	*	h(filesetSize-25600)
-0.0002108719	*	h(25600-filesetSize)
+0.0004302194	*	h(blockSize-20480)
-0.0004592948	*	h(20480-blockSize)
-0.06300488	*	h(readPercentage-25)
+0.1847696	*	h(25-readPercentage)
+9.93321	*	sequentialAccess

(b) Write Requests

Table 7.11.: MARS model.

coefficient is smaller than 10^{-9} for some parameters. This behavior increases the model size, especially if many levels interaction terms are involved. It is difficult to interpret the model: Although the coefficients seem to give a hint on the influence of each parameter this is not true: The influence depends on the range of the parameter. On the one hand a coefficient of 0.5 seems to produce a small influence if its parameter is the read percentage (ranges from 0% to 100%). On the other hand if its parameter is the block size (ranges from 4096 bytes to 32768 bytes), then this coefficient has a huge influence. This leads to the fact that it is impossible to judge on the influence of a parameter without knowing the parameter ranges. As explained above, linear models do not support factors. For this reason additional parameters have to be introduced which hinder interpretability. The interaction terms are difficult to imagine and to interpret. Nevertheless, as predicting a value using linear regression models only involves multiplication and a summation this calculation can be done easily even without a computer.

Table 7.11 and Table 7.12 contain the MARS models which are computed. They roughly show the same format as the linear models but there exist some differences: First, each model contains a selected set of terms. Not all terms must be included in the model but only those which are found valuable enough by the algorithm. Second, the hinge functions (also see Figure 4.4 for an explanation of the hinges) make the unique MARS ability of modeling partly linear functions possible. The coefficients here are comparable to those of the linear model. Like for the linear regression models the coefficients only show their true meaning when also looking at the parameters ranges. An additional difficulty lies in the interpretation of the hinge functions: These functions are partly zero and partly larger than zero. By multiplying them with a negative value they get smaller than zero. The term $h(filesetSize - 25600)$ does only influence the result if the file set size is bigger than 25 GB. As this areas of influence are overlaid it is difficult to get the overall structure of the model. This gets even more difficult if interaction terms are involved. To summarize one can say that MARS models have the same disadvantages as linear regression models plus the difficult interpretability of the hinge functions.

The M5 models which were created for this thesis are quite large: The write model contains more than 60 nodes and 34 leaves and the same number of linear regression models. As a graphical representation of this data is not possible, a smaller model is created for this section. This model is shrank by increasing the minimum amount of nodes which must be represented by a leaf. This means that the model depicted in Figure 7.13 is not the one which was discussed in the sections before. It is displayed here only to show how interpretable M5 models are. The full models which are generated in this thesis can be found in the Appendix C for reference purposes.

The M5 models are split into two parts: The regression tree and the linear models. Both parts are already discussed in separate: The regression tree is similar to the one generated

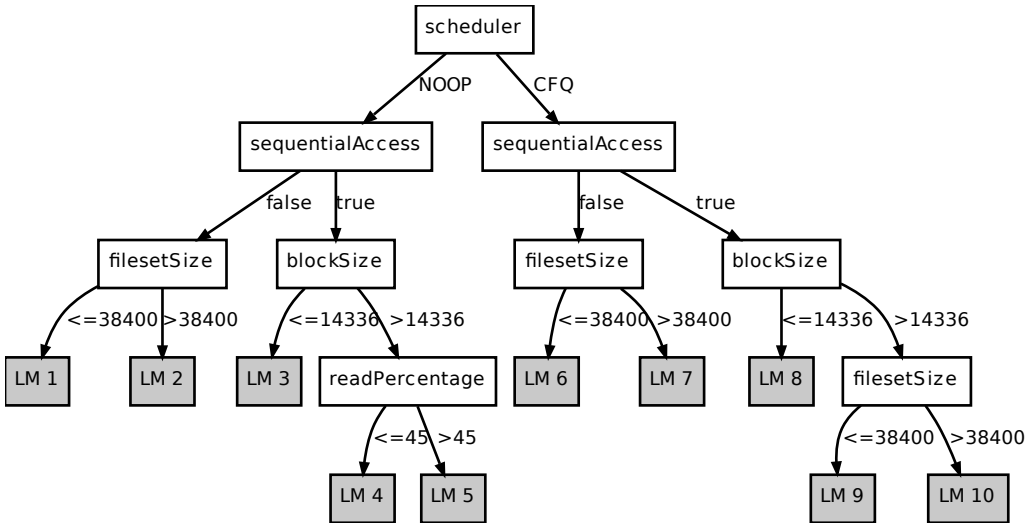
10.38197		
+0.0002565916	*	h(filesetSize-25600)
-0.0001162011	*	h(25600-filesetSize)
-0.0002583415	*	h(12288-blockSize)
-0.1084117	*	h(readPercentage-50)
+0.3249405	*	h(50-readPercentage)
+15.95859	*	sequentialAccess
-31.27198	*	schedulerNOOP*sequentialAccess
-0.0002109689	*	h(filesetSize-25600)*sequentialAccess
+0.001110919	*	h(blockSize-4096)*sequentialAccess
-0.000854648	*	schedulerNOOP*h(blockSize-12288)*sequentialAccess
+0.001082635	*	schedulerNOOP*h(12288-blockSize)*sequentialAccess

(a) Read Requests

15.11233		
-1.987135	*	schedulerNOOP
+8.297615e-05	*	h(filesetSize-25600)
-0.0002698644	*	h(25600-filesetSize)
+0.0002107494	*	h(blockSize-20480)
-0.0002541245	*	h(20480-blockSize)
-0.06300488	*	h(readPercentage-25)
+0.1847696	*	h(25-readPercentage)
+14.11624	*	sequentialAccess
-29.62917	*	schedulerNOOP*sequentialAccess
+0.0007417753	*	h(blockSize-4096)*sequentialAccess
-8.783138e-05	*	schedulerNOOP*h(filesetSize-25600)*sequentialAccess
+0.0002359698	*	schedulerNOOP*h(25600-filesetSize)*sequentialAccess
-0.0006056707	*	schedulerNOOP*h(blockSize-20480)*sequentialAccess
+0.0006628693	*	schedulerNOOP*h(20480-blockSize)*sequentialAccess

(b) Write Requests

Table 7.12.: Multidimensional MARS model.



(a) Regression Tree

Model	Scheduler	File Set Size	Block Size	Read Percentage	Seq. Access	Intercept
LM1	0.5508	0.0001	0.0002	0.1419	-0.3874	11.3216
LM2	0.5508	0.0002		-0.3145	-0.3874	21.9393
LM3	0.5508		0.0003	-0.0396	-0.3874	4.2814
LM4	0.5508		0.0003	-0.0296	-0.3874	4.0199
LM5	0.5508		0.0002	-0.0456	-0.3874	5.1961
LM6	0.5508	0.0001	0.0002	-0.1443	1.6657	12.0265
LM7	0.5508	0.0003		-0.3317	1.6657	21.4784
LM8	0.5508	0.0001	0.0013	-0.0319	1.6657	17.0547
LM9	0.5508	0.0002	0.0003	-0.2256	1.6657	42.8926
LM10	0.5508		0.0011	-0.043	1.6657	23.4465

(b) Linear Models

Figure 7.13.: M5 Model: The model depicted here is not the actual model which was created in the thesis. The real model has more than four times the size of this smaller model which does not show all branches of the original model. This model is depicted to show how interpretable the results are.

Criterion	Linear Regression	CART	MARS	M5
Generalization Ability	★★★	★	★★	★★★★
Model Generation Time	★★★★	★★★	★★	★
Prediction Time	★★★★	★★★★	★★	★
Interpretability	★★	★★★★	★	★
Simplicity of Algorithm	★★★★	★★	★★	★

Table 7.13.: Strengths and weaknesses of the different regression techniques and models. The comparison is done using a relative rating of one to four stars. The rating does only provide a relative ranking and does not account other existing modeling techniques. The "Simplicity of Algorithm" should be seen as the opposite of the complexity of the algorithm.

by the CART algorithm and the linear models are identical to linear regression models without interactions. The interpretation of the tree can be done easily as it shows the important decisions. The linear models are more difficult to interpret. It is easy to judge what has an influence by looking at the tree but quantifying and guessing the results involves the interpretation of the linear models which is difficult. The interpretability depends how much information is needed: If the regression tree provides enough information, it is easy. The interpretation gets more difficult if the linear models have to be analyzed. An additional point is the huge size of the M5 models: As explained above, the original model has 34 leaves and 34 linear models which make interpretation difficult simply because of the size.

The last metric used to compare the model is the *complexity of the algorithms and their implementation*. The main focus lies on the complexity of the algorithms as the implementation can be changed easily. Additionally, the implementations used for this thesis also include functionality which extends the original algorithms. All three algorithms are already explained in detail in Section 4.3.

The linear regression model algorithm using least square regression is a very simple algorithm. It can be transformed into simple matrix operations like multiplication and inversion of the input values. It only consists of a single step which makes the algorithm easy to understand and heavily reduces the complexity. Also the run time of the algorithm is constant for a given number of input samples and parameters.

The CART, MARS and M5 algorithms work in two steps: They first create a potentially overfitted model and then reduce its complexity. Both steps are aborted if a special condition is met. This makes the run time of the algorithm dependent on the actual data. Additionally the separation in two steps increases the complexity and also the runtime. The stop conditions for both runs must be chosen wisely, otherwise either a too small or an overfitted model results from the model generation. The M5 algorithm computes a lot of intermediate data which is later removed from the models: This includes the linear models which are computed for all nodes but are only used for the leaves. This creation and removal introduces another level of complexity in the algorithm. It should be kept in mind that much more complex algorithms exists for model generation. The CART and MARS algorithms were chosen because of their simple nature. This means that these considerations can only be seen relatively.

Table 7.13 contains a summary of the points discussed before. The table must not seen as an absolute assessment of the criteria but instead as a relative comparison of only these four regression techniques. Additionally, the generalization abilities and therefore the model quality is included in this comparison as it was discussed in the section before and must be included in the algorithm decisions.

Model	degree	nk	threshold	nprune
mars_multi	5	20	0.001	20
mars_var1	5	40	0.001	40
mars_var2	5	20	0.0001	20
mars_var3	5	40	0.0001	40

Table 7.14.: Different values for the MARS regression technique configuration parameters. The first column shows the name of the configuration.

The next section examines if the modeling techniques can perform better if they are differently configured.

7.2.3. Which configuration parameters of the regression techniques can improve the prediction results?

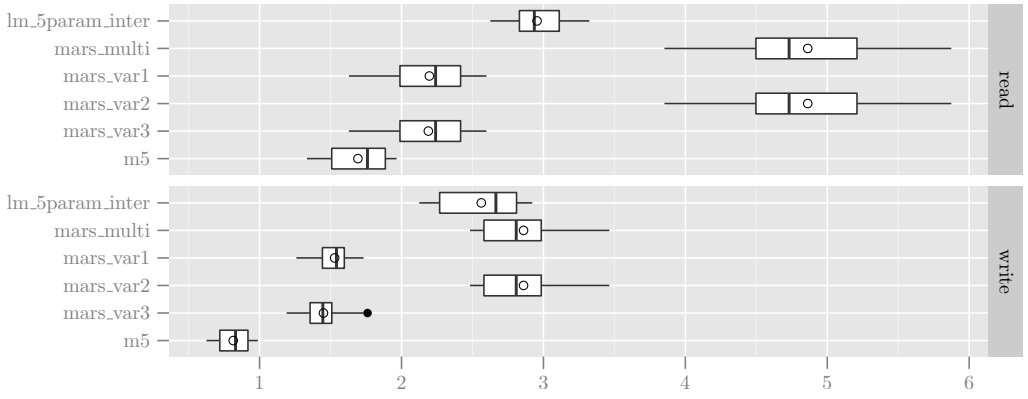
Both, the MARS and the CART algorithm can be configured using special parameters. The question, if setting these values to values other than their defaults improves the prediction results, is discussed here. The linear regression models are not discussed here as they do not feature any configuration options. Similar, the M5 model is not configurable in its currently used implementation and can therefore not be included in these discussions.

Section 4.3 already explained which configuration parameters exist and what these parameters control. For MARS, the *degree* parameter was already varied. As the ANOVA has shown, a bigger value increases the quality of the model heavily. For this reason the degree parameter is set fixed to a value of five. This does not mean that the MARS model is of degree five but only that the MARS algorithm can select to include interaction terms of up to five parameters. The *maximum number of terms in the forward step* (abbreviated `nk`) has a default value of 20. More nodes in the result of the forward step mean potentially more nodes after the pruning step. For this reason increasing the parameter increases the possibility of the model to adapt to details and smaller influences. Therefore the effects of setting this parameter to 40 is analyzed. This setting results in a model of potentially double the size of the original model. The *threshold for the forward step* is the second stopping condition for the forward step. Its default value is 0.001, which means that the forward step is stopped if the model has not improved by at least the threshold. Lowering this value lets the forward step include more terms in its output. For this reason a lower value of 0.0001 is tested. The *number of terms in the pruned model* (abbreviated `nprune`) are not limited by default, so their value is set to the same value as `nk`. This condition is left unchanged as setting this value higher does not make sense and setting it lower would hinder the ability to adapt to details.

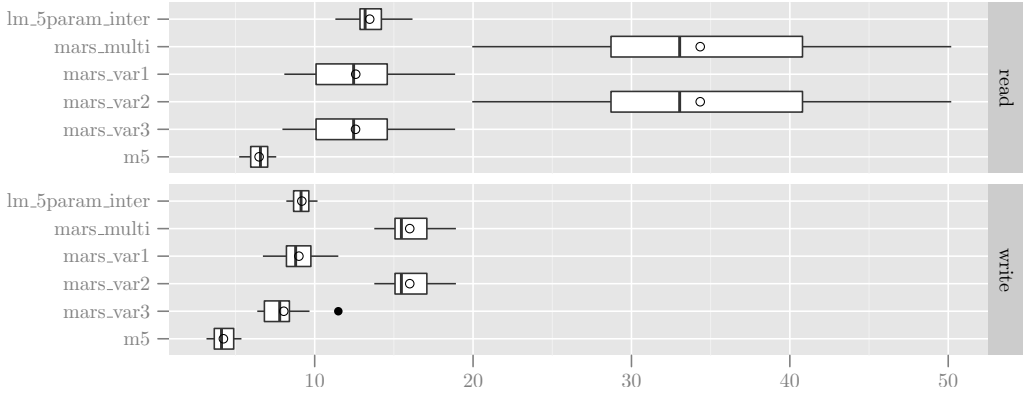
This considerations lead to the four configurations defined in Table 7.14. The first regression technique configuration is the default configuration which has already been analyzed in the previous sections. It is included here for comparison reasons. The models are referenced later using the names shown in the first column of the table.

These four MARS configurations are applied to the original sample set which is used for the nine models discussed before. The whole process of splitting the training data in training and test sets and the metric calculation was left unchanged.

Figure 7.14 contains the RMSE and MAPE values for the four models above together with the best linear regression model and the M5 model for comparison. The figure states clearly that changing the `threshold` parameter does not improve the models significantly. In fact the parameter changes nothing on the read model and improves very little on the write model. In contrast to that, the increase in the `nk` and `nprune` parameter improved the model heavily. This improvement makes the models even better than the best linear



(a) RMSE in ms



(b) MAPE in %

Figure 7.14.: Comparison of the quality of model constructed using different values for the configuration parameters of the MARS algorithm.

Model	minsplit	cp
cart	20	0.01
cart_var1	5	0.01
cart_var2	20	0.001
cart_var3	5	0.001

Table 7.15.: Different values for the CART regression technique configuration parameters. The first column shows the name of the configuration.

model. This is true for both, read and write models. With less than half the original value considering RMSE and only one third of the original value for MAPE, the improvements are obvious. The default value of only 20 terms seems to be not enough to model the complex behaviors of the storage system. A downside of the increase in the number of terms is the time needed for the construction of the model. This time is nearly doubled and also the time needed for the predictions is slightly increased.

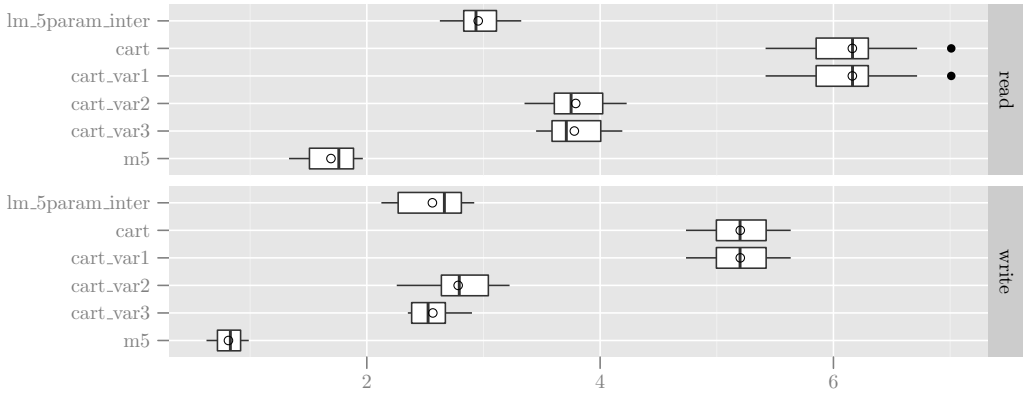
For CART, there exist two configuration parameters: The *minimum numbers of observations to try a split* (abbreviated `minsplit`) which has a default value of 20. Decreasing this value allows the algorithm to introduce more terms if needed. The second parameter is the *complexity parameter* (abbreviated `cp`). It is similar to the threshold parameter of MARS: A split of a node is not tried, if it does not improve the model by at least the value of the complexity parameter. The lower the value of this parameter, the more nodes are generated in the forward step. The default value for `cp` is 0.01. The pruning step of CART is fully automated and cannot be configured.

Based on these assumptions the four configurations depicted in Table 7.15 are defined. As above the first configuration contains only the default values and is already discussed in the previous sections.

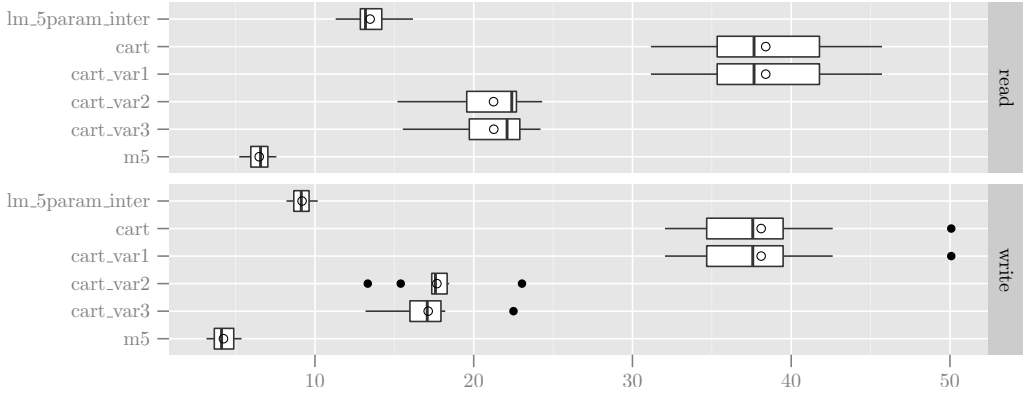
To compare these results the same procedure as above is repeated and results in the metrics depicted in Figure 7.15.

For this configuration, the `minsplit` parameter provides only very little improvement. In contrast to that, the raising of the `cp` parameter has a huge influence on the quality of the model. The improvement is not as large as with the `nk` parameter of MARS but still visible. Changing the `cp` parameter improves the CART model to the same level as a multidimensional MARS model using the default configuration. It is interesting to see that the `threshold` parameter did not influence the MARS model generation much but plays an important role for CART. This might result from the fact, that the threshold parameter is set one magnitude smaller as default for the MARS which might be sufficient. Another thing to notice is the fact that the creation time of the models does improve if the complexity parameter is set lower. In fact the time needed for the model construction is only half of the time needed for constructing the original model. It is not clear why this is the case as the pruning step of the CART algorithm has to prune the bigger model which should increase the time needed.

To summarize this question one can say that the choice of the configuration parameters of the regression techniques has a huge influence on the quality of the models. The default parameters have not always proven to be the best choice in this case of modeling storage performance using the system and the data collected for this thesis. Nevertheless it is not possible to define a generic good choice for the parameters. For example the quality of the models decreases if the values are set not strict enough: If increasing the `nk` parameter of the MARS model further, the models first gets better but at some point it starts to get worse again. This point is dependent on the data which is modeled. Therefore the parameters always have to be adapted to the needs. As the influence of the parameters



(a) RMSE in ms



(b) MAPE in %

Figure 7.15.: Comparison of the four different configurations for the CART algorithm. The unmodified CART model and the best linear regression model are included for comparison reasons.

depends on the data set, this is often not possible. If this adaption is not possible, the researcher must rely on the default values which were wisely chosen by the algorithm authors. Additionally, the finding of good parameters is time consuming. Furthermore, the experiments above show, that the models get better if their size is increased. An increase in the size also means more storage for the model and more time to create the models and predict new values. For example, the CART model `cart_var3` has thrice the size of the original CART model.

8. Conclusion

This chapter contains a summary of this thesis and and an outlook on future work.

8.1. Summary

This thesis presented an approach to the systematic analysis of storage performance. The approach was carried out by using separate research question defined in a Goal/Question/-Metric plan (compare Section 5.3). The analysis presented here is based on the systematic measurements which were carried out on an IBM System z and an IBM DS8700 storage controller.

The systematic gathering of the benchmark results is a time consuming job. It could be shown by comparing the separate results that a long enough benchmarking time for each experiment is advantageous. Due to the complex nature of the system and the amount of caching and logic involved, it is difficult to obtain stable results within a reasonable amount of time.

The results from the benchmarking were used to analyze the influence of different parameters on the storage performance. This was done using an analysis of variances (ANOVA) which lead to the conclusion that all five examined parameters have an influence on the variation of the storage performance. When analyzing the interactions between these parameters it also became clear that these interaction terms play an important role. This means that the parameters cannot be regarded alone but the interactions must be included in the examinations.

It was checked, if the benchmarking process at application layer can be extended to multiple virtualized machines. This involved the parallel execution of the benchmarks on two virtual machines and comparing the results to a single virtual machine run. The results were ambiguous: For pure workloads, where only read or write requests are issued, the measurements were comparable with a low error. When comparing the mixed workloads, the error increases heavily, that is, that the results of a benchmark of a single virtual machine cannot be simply translated to a multiple virtual machine scenario.

The results from the systematic benchmarking and the insights gained from the previous sections, especially from the analysis of parameters, was used to generate regression models. Nine regression models generated using four different techniques were focused: Linear regression, CART, MARS and M5. The nine models were trained using the systematic measurements and then tested to check their quality and generalization abilities.

The generalization abilities were divided into the interpolation abilities, the prediction of new samples from within the benchmarked ranges, and the extrapolation abilities, which includes samples from out of the benchmark range.

Interpolation using the storage performance prediction model works very well: The M5 algorithm shows to be the best performing algorithm with an error from as low as 4.24% (medium absolute percentage error) up to 10.39%, which still can be regarded a good result. The linear regression models also provided good results, despite their simple algorithm: Their absolute error can be as low as 9.19%. When increasing the complexity by building separate models for each scheduler and each access mode, the quality of the models increases significantly: The MARS algorithm can be used to generate models with an absolute error of 3.86%.

When looking at the extrapolation abilities of the regression models, the results are not that good: The models do not perform very well when used to predict values out of their training set ranges. The validation showed an error twice as high as for most of the models. Still, the linear regression models and the M5 algorithm performed best but yield to a relative error of more than 20%. This leads to the conclusion that, to prevent this high errors, more samples must be included in the modeling process to increase the range of the modeling.

The second part of the regression analysis involved the evaluation of the regression techniques: The models were compared with respect to their prediction quality, the time needed for the creation of the models and the prediction, the interpretability and the complexity of the algorithms. The M5 algorithm has shown an impressive quality for the predictions but has a slow model generation and prediction and is difficult to interpret. The linear regression can generate models very fast and is easier to interpret while still maintaining a good generalization ability. CART and MARS range in the midfield.

To extend the modeling discussion, the configuration parameters of the MARS and CART algorithms were analyzed. These parameters were left at their default values for the previous analyses. Their variation can improve the interpolation abilities of the models by up to 50%. The problem with these parameter changes is that the effects are not foreseeable: They are dependent on the actual benchmarked data and cannot be generalized. Additionally, the parameter changes which improve the quality of the models also increase the models size and the generation time.

To summarize the thesis: The generation of storage performance prediction models using regression techniques works well. These techniques provide a simple, yet versatile approach to the problem of predicting storage performance. When being provided with enough samples for the training process, the results can be improved further. The comparison between the regression techniques showed no absolute winner, therefore the trade offs between the techniques must be considered to find a good fit.

8.2. Future Work

Some topics, which could be a valuable contribution to the research community, were intentionally left out of scope in this thesis:

This thesis was limited to four regression techniques: Linear regression using least square, MARS, CART and M5. As the field of regression analysis is under constant improvement, it is highly possible that there are other techniques which provide better results. Their characteristics would have to be analyzed in details to see if these techniques are a good fit for storage performance modeling. In his master thesis, Faber [Fab11] has extensively discussed the advantages of genetic programming. These could fit well especially for more

complex models and scenarios. Another technology which could be used are artificial neural network models as used by Kundu et al. [KRDZ10]. Furthermore there exists a number of regression tree algorithms similar to CART and M5. The integration of new regression techniques in the existing Analysis Library developed in this thesis can easily be done if the algorithms are implemented in R.

Benchmarking additional parameter configurations could produce better results and provide a better insight to both, storage performance modeling and the storage system structure: The thesis limited the operations to read and write. Other operations, which modify the file system were not considered. The effect of the system load which includes the thread count has not been included in the considerations. Operating system caches were not included in the models. Some parameters were coupled and could be decoupled: The read and write block size could be benchmarked separately and the combination of sequential reads and random writes (and vice–versa) could be evaluated.

Although this thesis used a heavily virtualized system for the benchmarking, many questions remain open in this context: A goal could be to include multiple virtual machines and their configuration in the modeling. This would mean to include one workload parameter set for each virtual machine involved. The predictions could include the run response time of the requests of each of the machines separately. Because this would heavily increase the benchmark work, an approach similar to the one proposed by Westermann et al. and Kundu et al. [WHKF12, KRDZ10] is helpful: They proposed an iterative approach where the models are refined as needed. This reduces the time required for the benchmarking.

Mi et al. [MCCS08] show a way to include burstiness in regression models. This approach could increase the quality of the models and reduce the benchmarking time required to create accurate models.

Appendix

A. ANOVA Including All Interaction Terms

In Section 6.2, an ANOVA is conducted using no interactions and another ANOVA is executed using interactions of two parameters. The following Table A.1 and Table A.2 present ANOVAs containing all interaction terms possible. The tables are left out of Section 6.2 due their length and their missing additional value for the interpretation.

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
filesetSize	4	129805.09	9.36	32451.27	6942.70	0.0000
blockSize	7	67227.21	4.85	9603.89	2054.68	0.0000
sequentialAccess	1	64252.95	4.63	64252.95	13746.43	0.0000
scheduler	1	396445.48	28.58	396445.48	84816.51	0.0000
readPercentage	5	91672.27	6.61	18334.45	3922.52	0.0000
filesetSize:blockSize	28	300.09	0.02	10.72	2.29	0.0001
filesetSize:sequentialAccess	4	48776.42	3.52	12194.10	2608.84	0.0000
filesetSize:scheduler	4	7805.86	0.56	1951.47	417.50	0.0000
filesetSize:readPercentage	20	4569.28	0.33	228.46	48.88	0.0000
blockSize:sequentialAccess	7	43219.46	3.12	6174.21	1320.93	0.0000
blockSize:scheduler	7	15613.25	1.13	2230.46	477.19	0.0000
blockSize:readPercentage	35	608.33	0.04	17.38	3.72	0.0000
sequentialAccess:scheduler	1	388654.37	28.02	388654.37	83149.67	0.0000
sequentialAccess:readPercentage	5	23591.17	1.70	4718.23	1009.43	0.0000
scheduler:readPercentage	5	1445.42	0.10	289.08	61.85	0.0000
filesetSize:blockSize:sequentialAccess	28	6746.33	0.49	240.94	51.55	0.0000
filesetSize:blockSize:scheduler	28	1341.73	0.10	47.92	10.25	0.0000
filesetSize:blockSize:readPercentage	140	1902.92	0.14	13.59	2.91	0.0000
filesetSize:sequentialAccess:scheduler	4	6682.47	0.48	1670.62	357.42	0.0000
filesetSize:sequentialAccess:readPercentage	20	23531.29	1.70	1176.56	251.72	0.0000
filesetSize:scheduler:readPercentage	20	2210.41	0.16	110.52	23.65	0.0000
blockSize:sequentialAccess:scheduler	7	29548.79	2.13	4221.26	903.11	0.0000
blockSize:sequentialAccess:readPercentage	35	1250.36	0.09	35.72	7.64	0.0000
blockSize:scheduler:readPercentage	35	453.85	0.03	12.97	2.77	0.0000
sequentialAccess:scheduler:readPercentage	5	760.35	0.05	152.07	32.53	0.0000
filesetSize:blockSize:sequentialAccess:scheduler	28	3759.00	0.27	134.25	28.72	0.0000
filesetSize:blockSize:sequentialAccess:readPercentage	140	1344.75	0.10	9.61	2.05	0.0000
filesetSize:blockSize:scheduler:readPercentage	140	1195.82	0.09	8.54	1.83	0.0000
filesetSize:sequentialAccess:scheduler:readPercentage	20	2730.12	0.20	136.51	29.20	0.0000
blockSize:sequentialAccess:scheduler:readPercentage	35	624.40	0.05	17.84	3.82	0.0000
filesetSize:blockSize:sequentialAccess:scheduler:readPercentage	140	937.53	0.07	6.70	1.43	0.0008
Residuals	3840	17948.75	1.29	4.67		

Table A.1.: ANOVA containing all interactions for read requests.

	Df	Sum Sq	rSum Sq	Mean Sq	F value	Pr(>F)
filesetSize	4	55241.76	5.58	13810.44	11495.84	0.0000
blockSize	7	84777.97	8.56	12111.14	10081.34	0.0000
sequentialAccess	1	118402.39	11.96	118402.39	98558.38	0.0000
scheduler	1	338859.54	34.22	338859.54	282067.34	0.0000
readPercentage	5	30199.64	3.05	6039.93	5027.65	0.0000
filesetSize:blockSize	28	288.46	0.03	10.30	8.58	0.0000
filesetSize:sequentialAccess	4	8163.24	0.82	2040.81	1698.77	0.0000
filesetSize:scheduler	4	11948.33	1.21	2987.08	2486.45	0.0000
filesetSize:readPercentage	20	1232.97	0.12	61.65	51.32	0.0000
blockSize:sequentialAccess	7	18968.06	1.92	2709.72	2255.58	0.0000
blockSize:scheduler	7	13985.36	1.41	1997.91	1663.06	0.0000
blockSize:readPercentage	35	553.49	0.06	15.81	13.16	0.0000
sequentialAccess:scheduler	1	263456.40	26.60	263456.40	219301.62	0.0000
sequentialAccess:readPercentage	5	6396.13	0.65	1279.23	1064.83	0.0000
scheduler:readPercentage	5	899.03	0.09	179.81	149.67	0.0000
filesetSize:blockSize:sequentialAccess	28	2039.96	0.21	72.86	60.65	0.0000
filesetSize:blockSize:scheduler	28	1129.01	0.11	40.32	33.56	0.0000
filesetSize:blockSize:readPercentage	140	1064.83	0.11	7.61	6.33	0.0000
filesetSize:sequentialAccess:scheduler	4	5718.56	0.58	1429.64	1190.04	0.0000
filesetSize:sequentialAccess:readPercentage	20	6113.71	0.62	305.69	254.45	0.0000
filesetSize:scheduler:readPercentage	20	2186.09	0.22	109.30	90.99	0.0000
blockSize:sequentialAccess:scheduler	7	8246.76	0.83	1178.11	980.66	0.0000
blockSize:sequentialAccess:readPercentage	35	450.70	0.05	12.88	10.72	0.0000
blockSize:scheduler:readPercentage	35	1255.05	0.13	35.86	29.85	0.0000
sequentialAccess:scheduler:readPercentage	5	180.80	0.02	36.16	30.10	0.0000
filesetSize:blockSize:sequentialAccess:scheduler	28	441.71	0.04	15.78	13.13	0.0000
filesetSize:blockSize:sequentialAccess:readPercentage	140	368.63	0.04	2.63	2.19	0.0000
filesetSize:blockSize:scheduler:readPercentage	140	791.94	0.08	5.66	4.71	0.0000
filesetSize:sequentialAccess:scheduler:readPercentage	20	1859.33	0.19	92.97	77.39	0.0000
blockSize:sequentialAccess:scheduler:readPercentage	35	216.34	0.02	6.18	5.15	0.0000
filesetSize:blockSize:sequentialAccess:scheduler:readPercentage	140	270.66	0.03	1.93	1.61	0.0000
Residuals	3840	4613.16	0.47	1.20		

Table A.2.: ANOVA for write response times containing all interactions.

B. Comparison of All Models

This section contains four figures: Figure B.1 and Figure B.2 contain the ten–fold cross–validation of all models which were evaluated in this thesis. Each of the box plots contains the results of one of the ten folds. For the interpretation of the plots see the appropriate chapters. The next two figures contain the comparison of the duration of model creation (Figure B.3) and sample prediction (Figure B.4) for all models which were evaluated in this thesis. The underlying data can be found in two tables: The results from the cross–validation in Table B.3 and the timing results in Table B.4.

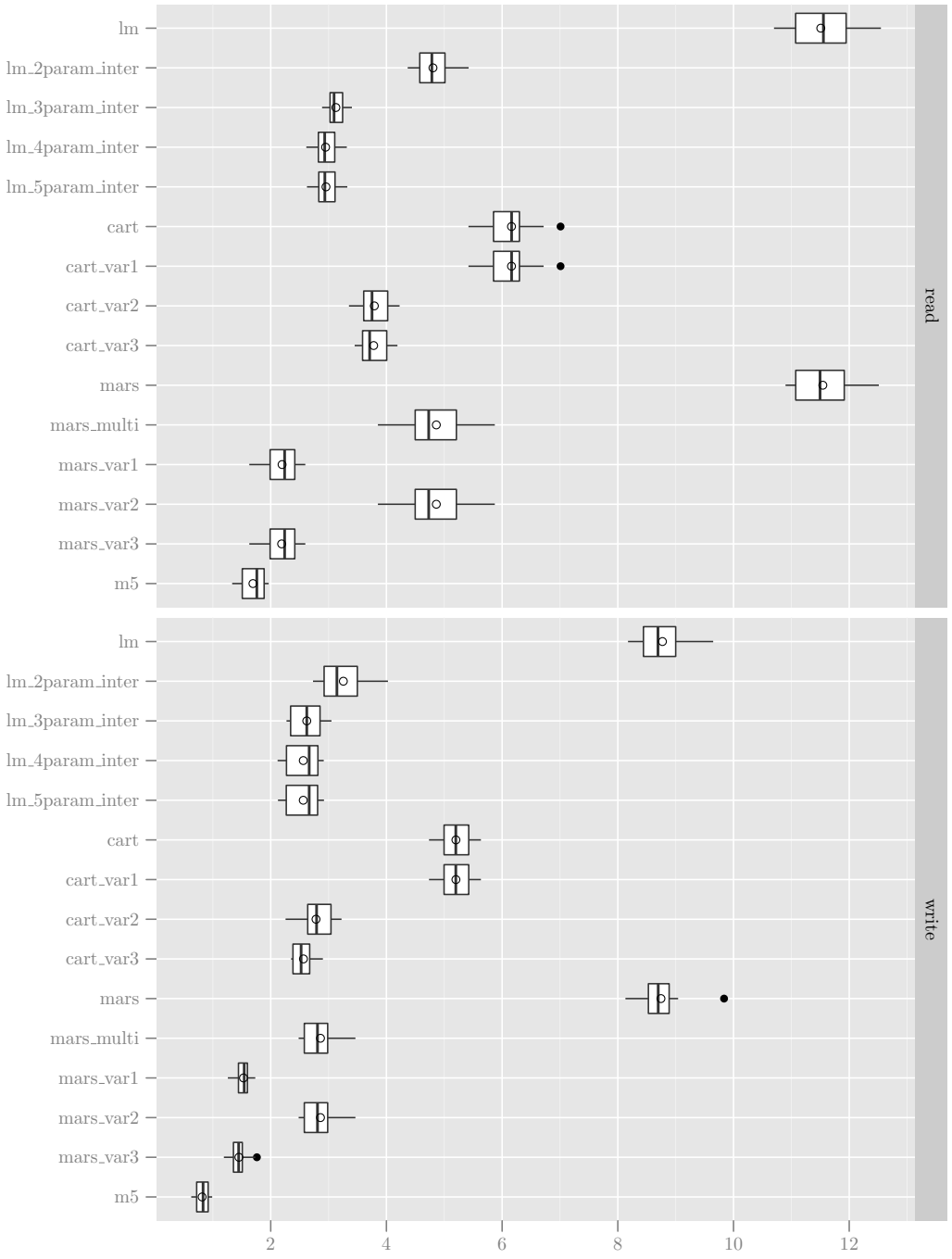


Figure B.1.: Comparison of all 15 models which were generated in this thesis using RMSE when using 10–fold cross–validation to test the interpretation abilities.

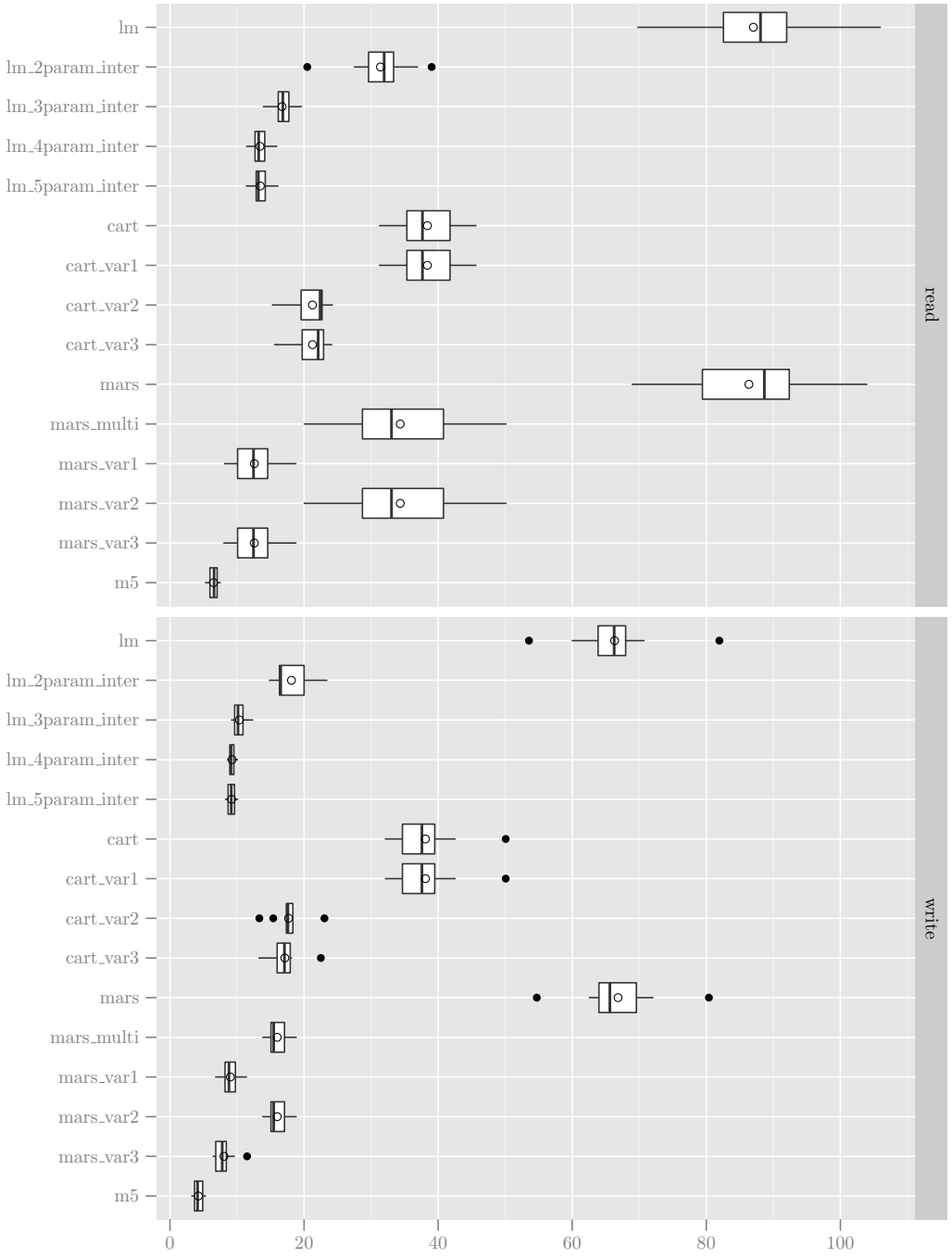


Figure B.2.: Comparison of the interpretation abilities of all 15 models when using 10–fold cross–validation and using MAPE as metric.

Model	Read Model		Write Model	
	RMSE (ms)	MAPE (%)	RMSE (ms)	MAPE (%)
lm	11.51	87.02	8.77	66.32
lm_2param_inter	4.80	31.40	3.26	18.12
lm_3param_inter	3.13	16.72	2.62	10.35
lm_4param_inter	2.95	13.43	2.56	9.26
lm_5param_inter	2.96	13.47	2.56	9.19
cart	6.16	38.40	5.20	38.11
cart_var1	6.16	38.40	5.20	38.11
cart_var2	3.79	21.25	2.78	17.68
cart_var3	3.78	21.26	2.57	17.13
mars	11.54	86.36	8.75	66.84
mars_multi	4.86	34.34	2.86	16.00
mars_var1	2.20	12.59	1.53	9.00
mars_var2	4.86	34.34	2.86	16.00
mars_var3	2.19	12.58	1.45	8.05
m5	1.69	6.49	0.81	4.24

Table B.3.: Comparison of the interpolation and generalization abilities of all 15 models, benchmarked using 10–fold cross–validation.

Model	Modeling		Prediction	
	Read (s)	Write (s)	Read (s)	Write (s)
lm	0.00500	0.00500	0.00167	0.00115
lm_2param_inter	0.00800	0.00800	0.00135	0.00135
lm_3param_inter	0.01200	0.01200	0.00167	0.00156
lm_4param_inter	0.01400	0.01500	0.00188	0.00167
lm_5param_inter	0.01500	0.01400	0.00177	0.00177
cart	0.02500	0.02500	0.00115	0.00104
cart_var1	0.02500	0.02500	0.00115	0.00104
cart_var2	0.02600	0.02600	0.00125	0.00125
cart_var3	0.02700	0.02600	0.00125	0.00115
mars	0.04400	0.04400	0.02083	0.02042
mars_multi	0.05800	0.05600	0.02125	0.02146
mars_var1	0.11100	0.10500	0.02542	0.02385
mars_var2	0.05700	0.05500	0.02198	0.02177
mars_var3	0.11100	0.10600	0.02521	0.02396
m5	0.14600	0.18100	0.05052	0.05156

Table B.4.: Comparison of the time needed for the model creation ("Modeling") and for the prediction of 1000 samples ("Prediction"). The time needed for both, the read and the write models are depicted for each of the 15 models.

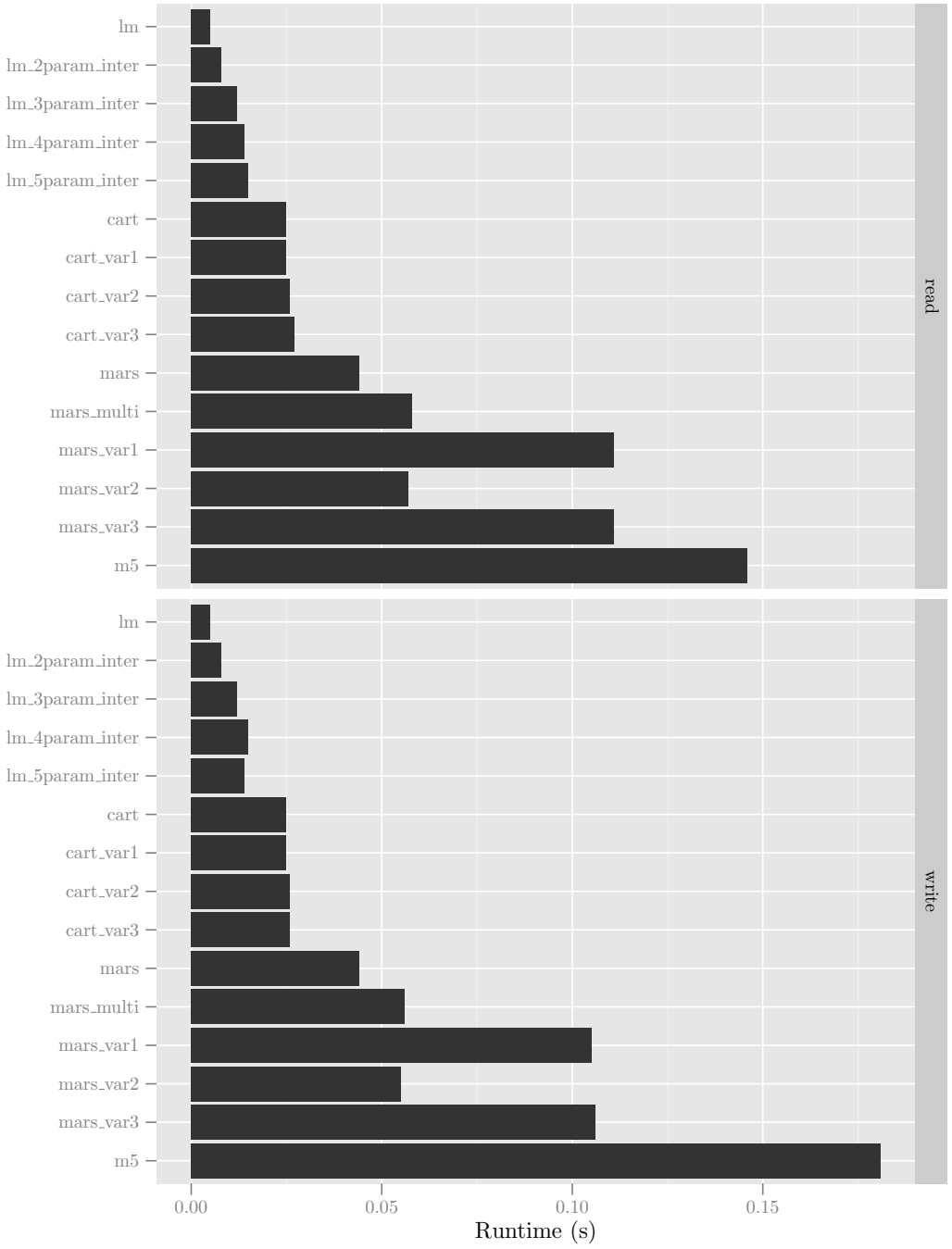


Figure B.3.: Comparison of all 15 models which were generated in this thesis. Compared by time needed for model generation.

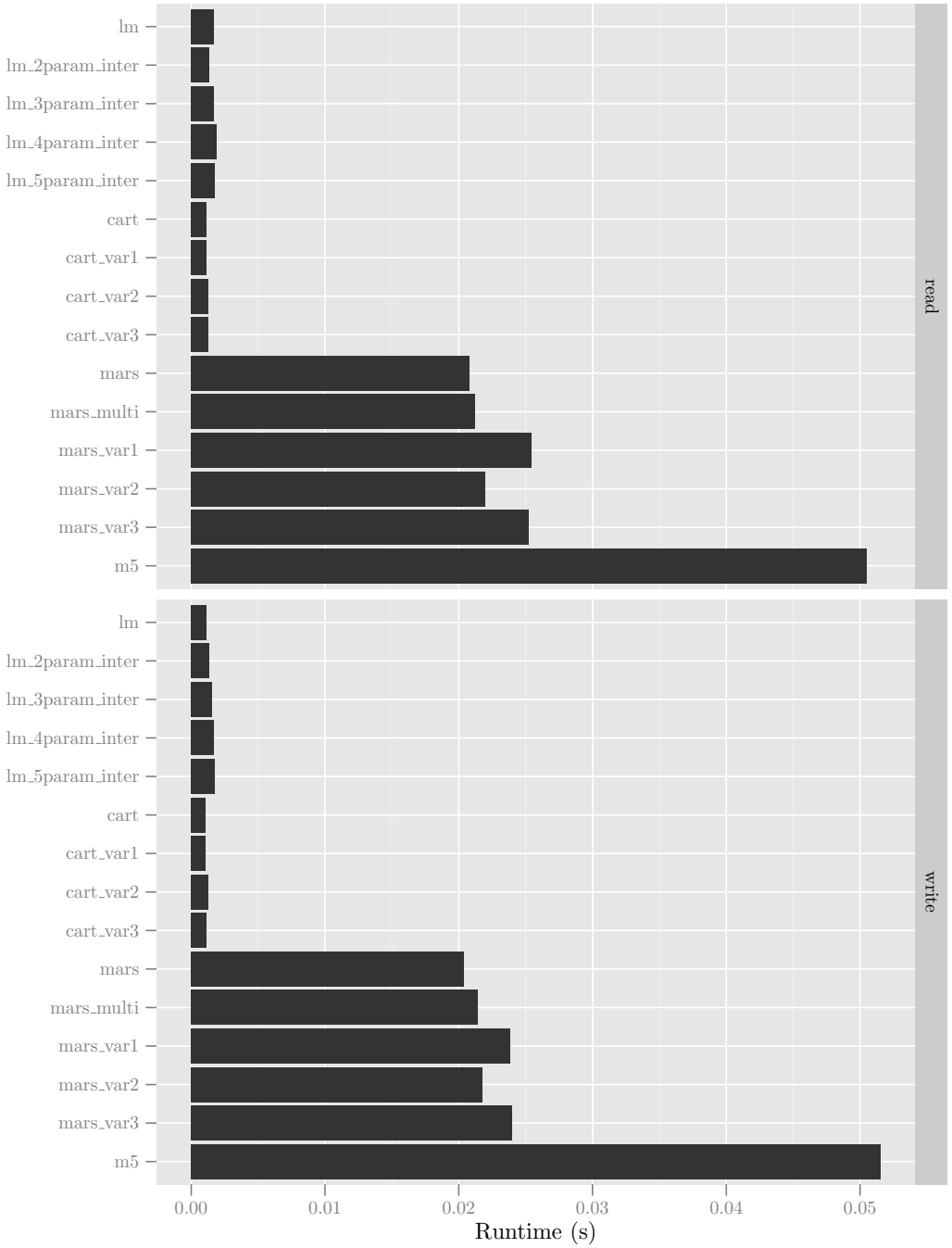


Figure B.4.: Comparison of all 15 models which were generated in this thesis. Compared by time needed for prediction of 1000 samples.

C. M5 Model

The following two texts are a textual representation of the M5 models which were created in this thesis. As explained, the model is so large that a graphical representation was not possible anymore:

C.1. Read Model

```
Rule 1: [60 cases, mean 4.071682, range 2.186175 to 5.945354, est err 0.208076]

  if
  schedulerNOOP > 0
  blockSize <= 16384
  readPercentage > 50
  sequentialAccess > 0
  then
  outcome = 2.249239 + 0.00023 blockSize - 0.007 readPercentage

Rule 2: [60 cases, mean 5.400255, range 2.999963 to 10.08509, est err 0.377734]

  if
  schedulerNOOP > 0
  blockSize <= 16384
  readPercentage <= 50
  sequentialAccess > 0
  then
  outcome = 4.230993 + 0.00031 blockSize - 0.054 readPercentage

Rule 3: [64 cases, mean 5.753605, range 2.18695 to 8.427136, est err 0.301927]

  if
  filesetSize <= 25600
  readPercentage > 60
  sequentialAccess <= 0
  then
  outcome = 2.237216 + 0.00018 blockSize + 1.7e-05 filesetSize

Rule 4: [32 cases, mean 5.779727, range 2.18695 to 8.427136, est err 0.302356]

  if
  schedulerNOOP > 0
  filesetSize <= 25600
  readPercentage > 60
  sequentialAccess <= 0
  then
  outcome = 3.186848 + 0.00019 blockSize + 1.3e-05 filesetSize
            - 0.012 readPercentage

Rule 5: [60 cases, mean 7.529825, range 6.058173 to 9.877037, est err 0.304938]

  if
  schedulerNOOP > 0
  blockSize > 16384
  readPercentage > 50
  sequentialAccess > 0
  then
  outcome = 3.93149 + 0.00019 blockSize - 0.018 readPercentage

Rule 6: [48 cases, mean 8.935019, range 3.280741 to 18.30055, est err 0.896683]

  if
  schedulerNOOP > 0
  filesetSize <= 1024
  readPercentage <= 50
  then
  outcome = 10.177727 - 0.221 readPercentage + 0.00038 blockSize

Rule 7: [64 cases, mean 8.977553, range 2.955563 to 19.05015, est err 1.456868]

  if
  filesetSize <= 25600
  blockSize <= 16384
  readPercentage <= 60
  sequentialAccess <= 0
  then
  outcome = 12.25084 + 0.00028 filesetSize - 0.205 readPercentage
            + 0.00023 blockSize

Rule 8: [48 cases, mean 10.208871, range 7.007518 to 14.74364, est err 0.254249]

  if
  schedulerNOOP > 0
  filesetSize > 1024
  blockSize > 16384
  readPercentage <= 50
  sequentialAccess > 0
  then
  outcome = 5.841694 + 0.0003 blockSize - 0.113 readPercentage
            + 1.1e-05 filesetSize

Rule 9: [64 cases, mean 10.753127, range 2.955563 to 21.09304, est err 1.289743]

  if
  schedulerNOOP > 0
  filesetSize <= 25600
```

```
  readPercentage <= 60
  sequentialAccess <= 0
  then
  outcome = 13.712694 + 0.000201 filesetSize - 0.21 readPercentage
            + 0.00026 blockSize - 1.1 schedulerNOOP + 0.5 sequentialAccess

Rule 10: [64 cases, mean 12.357505, range 6.706121 to 21.09304, est err 1.123855]

  if
  filesetSize <= 25600
  blockSize > 16384
  readPercentage <= 60
  sequentialAccess <= 0
  then
  outcome = 16.076314 - 0.228 readPercentage + 7e-05 filesetSize
            + 0.00018 blockSize

Rule 11: [48 cases, mean 14.328711, range 3.179194 to 25.22037, est err 1.084740]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  readPercentage > 60
  sequentialAccess <= 0
  then
  outcome = 9.696136 + 0.000243 filesetSize - 0.17 readPercentage
            + 1.5 sequentialAccess + 3e-05 blockSize

Rule 12: [72 cases, mean 16.094955, range 3.612804 to 27.41615, est err 0.846966]

  if
  schedulerNOOP > 0
  filesetSize > 25600
  readPercentage > 50
  sequentialAccess <= 0
  then
  outcome = 10.785956 + 0.000241 filesetSize - 0.192 readPercentage
            + 0.0001 blockSize

Rule 13: [48 cases, mean 21.867498, range 12.13931 to 32.26341, est err 0.971340]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  readPercentage > 40
  readPercentage <= 60
  sequentialAccess <= 0
  then
  outcome = 24.912634 + 0.000252 filesetSize - 0.366 readPercentage
            - 0.00011 blockSize

Rule 14: [12 cases, mean 23.787912, range 17.12643 to 31.60984, est err 0.617917]

  if
  schedulerNOOP <= 0
  filesetSize <= 1024
  blockSize <= 16384
  readPercentage > 50
  sequentialAccess > 0
  then
  outcome = 30.941452 + 0.00059 blockSize - 0.17 readPercentage

Rule 15: [12 cases, mean 29.849649, range 22.44754 to 37.83307, est err 0.735931]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  filesetSize <= 25600
  blockSize <= 16384
  readPercentage > 50
  sequentialAccess > 0
  then
  outcome = 22.853144 + 0.00107 blockSize - 0.053 readPercentage

Rule 16: [72 cases, mean 30.332626, range 16.81138 to 48.10731, est err 1.382687]

  if
  schedulerNOOP > 0
  filesetSize > 25600
  readPercentage <= 50
  sequentialAccess <= 0
  then
  outcome = 30.264563 - 0.574 readPercentage + 0.000267 filesetSize
            + 0.0001 blockSize

Rule 17: [24 cases, mean 31.847515, range 19.93114 to 47.4202, est err 1.327085]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  blockSize > 16384
  readPercentage <= 40
  sequentialAccess <= 0
  then
  outcome = 31.772987 - 0.675 readPercentage + 0.000287 filesetSize

Rule 18: [12 cases, mean 32.867348, range 24.33015 to 42.49104, est err 0.596964]

  if
  schedulerNOOP <= 0
```

```
  filesetSize > 1024
  filesetSize <= 25600
  blockSize <= 16384
  readPercentage <= 50
  sequentialAccess > 0
  then
  outcome = 20.875949 + 0.00135 blockSize - 0.047 readPercentage

Rule 19: [36 cases, mean 34.725925, range 23.49752 to 45.69067, est err 0.688417]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  blockSize <= 16384
  readPercentage > 50
  sequentialAccess > 0
  then
  outcome = 19.700927 + 0.00153 blockSize + 3e-05 filesetSize
            - 0.04 readPercentage

Rule 20: [36 cases, mean 36.340027, range 25.49491 to 49.33521, est err 0.926318]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  blockSize <= 16384
  readPercentage <= 50
  sequentialAccess > 0
  then
  outcome = 21.875018 + 0.00164 blockSize - 0.059 readPercentage

Rule 21: [24 cases, mean 37.363880, range 23.32743 to 59.78062, est err 2.429394]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  blockSize <= 16384
  readPercentage <= 40
  sequentialAccess <= 0
  then
  outcome = 34.805767 - 0.799 readPercentage + 0.000368 filesetSize
            + 0.5 sequentialAccess

Rule 22: [32 cases, mean 40.512684, range 24.93643 to 55.38382, est err 1.342345]

  if
  schedulerNOOP <= 0
  filesetSize <= 25600
  blockSize > 16384
  readPercentage > 40
  sequentialAccess > 0
  then
  outcome = 31.667796 + 0.000466 filesetSize + 0.00068 blockSize
            - 0.215 readPercentage

Rule 23: [24 cases, mean 42.319393, range 25.67785 to 64.67324, est err 1.812921]

  if
  schedulerNOOP <= 0
  filesetSize <= 1024
  readPercentage <= 50
  sequentialAccess > 0
  then
  outcome = 45.691879 - 0.506 readPercentage + 0.00089 blockSize

Rule 24: [16 cases, mean 52.540298, range 42.65955 to 64.67324, est err 3.034891]

  if
  schedulerNOOP <= 0
  filesetSize <= 25600
  blockSize > 16384
  readPercentage <= 40
  sequentialAccess > 0
  then
  outcome = 43.369215 - 0.462 readPercentage + 0.00092 blockSize

Rule 25: [32 cases, mean 57.396091, range 40.46554 to 68.46743, est err 1.882137]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  filesetSize <= 76800
  blockSize > 16384
  readPercentage > 40
  sequentialAccess > 0
  then
  outcome = 26.061333 + 0.00121 blockSize + 8.8e-05 filesetSize
            - 0.088 readPercentage

Rule 26: [24 cases, mean 58.428070, range 48.25661 to 67.93386, est err 1.221291]

  if
  schedulerNOOP <= 0
  filesetSize > 76800
  blockSize > 16384
  sequentialAccess > 0
  then
  outcome = 26.41159 + 0.00125 blockSize - 0.033 readPercentage

Rule 27: [24 cases, mean 59.011002, range 42.1826 to 70.36684, est err 4.958269]
```

```
  if
  schedulerNOOP <= 0
  filesetSize > 25600
  blockSize > 16384
  readPercentage <= 40
  sequentialAccess > 0
  then
outcome = 44.936104 + 0.00105 blockSize - 0.394 readPercentage
```

C.2. Write Model

```
Rule 1: [36 cases, mean 4.545806, range 2.766769 to 8.540091, est err 0.459925]

  if
  filesetSize <= 1024
  blockSize <= 12288
  sequentialAccess <= 0
  then
  outcome = 3.211877 + 0.00036 blockSize + 2.8 sequentialAccess
            - 0.039 readPercentage + 1.4e-05 filesetSize

Rule 2: [20 cases, mean 4.999155, range 2.766769 to 8.169867, est err 0.246163]

  if
  schedulerNOOP > 0
  filesetSize <= 1024
  blockSize <= 20480
  readPercentage > 25
  sequentialAccess <= 0
  then
  outcome = 4.361609 + 0.00026 blockSize - 0.036 readPercentage
            - 0.6 schedulerNOOP + 0.4 sequentialAccess

Rule 3: [25 cases, mean 5.259614, range 2.768775 to 9.476368, est err 0.343393]

  if
  schedulerNOOP > 0
  filesetSize <= 1024
  blockSize <= 20480
  readPercentage > 0
  sequentialAccess > 0
  then
  outcome = 4.562247 + 0.00027 blockSize - 0.044 readPercentage
            - 0.5 sequentialAccess + 6e-06 filesetSize

Rule 4: [80 cases, mean 5.514467, range 2.941291 to 9.416635, est err 0.273063]

  if
  schedulerNOOP > 0
  filesetSize > 1024
  blockSize <= 20480
  readPercentage > 25
  sequentialAccess > 0
  then
  outcome = 4.631451 + 0.00028 blockSize - 0.046 readPercentage

Rule 5: [32 cases, mean 6.871717, range 2.850726 to 12.34166, est err 0.352914]

  if
  schedulerNOOP <= 0
  filesetSize <= 1024
  readPercentage > 25
  sequentialAccess <= 0
  then
  outcome = 4.30897 + 0.00028 blockSize - 0.048 readPercentage

Rule 6: [64 cases, mean 9.173804, range 3.365226 to 14.7646, est err 0.505325]

  if
  schedulerNOOP > 0
  filesetSize > 1024
  filesetSize <= 51200
  readPercentage > 25
  sequentialAccess <= 0
  then
  outcome = 11.029722 - 0.145 readPercentage + 9.5e-05 filesetSize
            + 0.00014 blockSize

Rule 7: [40 cases, mean 9.483462, range 3.416123 to 15.79906, est err 0.328182]

  if
  schedulerNOOP > 0
  readPercentage > 0
  readPercentage <= 25
  sequentialAccess > 0
  then
  outcome = 2.116236 + 0.00041 blockSize

Rule 8: [30 cases, mean 9.870280, range 6.767488 to 14.19027, est err 0.274691]

  if
  schedulerNOOP > 0
  filesetSize <= 1024
  blockSize > 20480
  readPercentage > 0
  then
```

```
  outcome = 6.428383 + 0.00029 blockSize - 0.099 readPercentage

Rule 9: [15 cases, mean 9.932889, range 3.439524 to 14.48971, est err 1.250289]

  if
  schedulerNOOP > 0
  filesetSize <= 51200
  blockSize <= 20480
  readPercentage > 0
  readPercentage <= 25
  sequentialAccess <= 0
  then
  outcome = 2.430456 + 0.000111 filesetSize + 0.00034 blockSize

Rule 10: [30 cases, mean 10.671844, range 6.97946 to 15.72283, est err 0.353073]

  if
  schedulerNOOP > 0
  filesetSize > 51200
  blockSize > 20480
  readPercentage > 0
  sequentialAccess > 0
  then
  outcome = 7.748855 - 0.121 readPercentage + 0.00031 blockSize

Rule 11: [30 cases, mean 11.151483, range 7.348397 to 15.79906, est err 0.237698]

  if
  schedulerNOOP > 0
  filesetSize > 1024
  filesetSize <= 51200
  blockSize > 20480
  readPercentage > 0
  sequentialAccess > 0
  then
  outcome = 7.724286 + 0.00033 blockSize - 0.113 readPercentage
            - 1e-05 filesetSize

Rule 12: [40 cases, mean 11.656739, range 4.239166 to 19.18966, est err 0.048223]

  if
  schedulerNOOP > 0
  readPercentage <= 0
  sequentialAccess > 0
  then
  outcome = 2.085253 + 0.00052 blockSize

Rule 13: [30 cases, mean 11.675081, range 7.331079 to 16.95288, est err 0.686488]

  if
  schedulerNOOP > 0
  filesetSize > 1024
  filesetSize <= 51200
  blockSize > 20480
  readPercentage > 0
  sequentialAccess <= 0
  then
  outcome = 13.046082 - 0.153 readPercentage + 0.00017 blockSize
            + 3.5e-05 filesetSize

Rule 14: [20 cases, mean 11.877479, range 3.490562 to 19.67571, est err 0.617486]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  blockSize <= 4096
  readPercentage > 0
  sequentialAccess <= 0
  then
  outcome = 10.516692 + 0.000131 filesetSize - 0.138 readPercentage

Rule 15: [70 cases, mean 12.212640, range 4.426095 to 18.57304, est err 0.842507]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  filesetSize <= 51200
  blockSize > 4096
  readPercentage > 0
  sequentialAccess <= 0
  then
  outcome = 9.394175 - 0.111 readPercentage + 7.3e-05 filesetSize
            + 0.00025 blockSize

Rule 16: [20 cases, mean 12.908865, range 7.759267 to 19.10088, est err 0.443105]

  if
  filesetSize <= 1024
  blockSize > 12288
  readPercentage <= 25
  sequentialAccess <= 0
  then
  outcome = 3.54713 + 0.00046 blockSize - 0.155 readPercentage

Rule 17: [16 cases, mean 13.591073, range 9.359589 to 18.57304, est err 0.475694]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  filesetSize <= 25600
```

```
  blockSize > 16384
  readPercentage > 0
  readPercentage <= 60
  sequentialAccess <= 0
  then
  outcome = 14.458815 - 0.164 readPercentage + 0.00024 blockSize

Rule 18: [50 cases, mean 15.032749, range 9.201373 to 19.66216, est err 0.588060]

  if
  schedulerNOOP > 0
  filesetSize > 51200
  blockSize <= 20480
  readPercentage > 0
  sequentialAccess <= 0
  then
  outcome = 6.850477 + 0.000124 filesetSize - 0.09 readPercentage
            + 0.00013 blockSize

Rule 19: [30 cases, mean 15.482719, range 10.70855 to 19.24647, est err 1.006962]

  if
  schedulerNOOP <= 0
  filesetSize > 51200
  blockSize > 4096
  blockSize <= 16384
  readPercentage > 0
  sequentialAccess <= 0
  then
  outcome = 3.362062 + 0.00011 filesetSize + 0.0003 blockSize
            - 0.025 readPercentage + 0.4 sequentialAccess

Rule 20: [16 cases, mean 15.962940, range 4.239478 to 24.78856, est err 1.095467]

  if
  schedulerNOOP > 0
  filesetSize <= 25600
  readPercentage <= 0
  sequentialAccess <= 0
  then
  outcome = 3.200554 + 0.000356 filesetSize + 0.00044 blockSize

Rule 21: [30 cases, mean 16.594831, range 11.45163 to 21.21955, est err 0.532943]

  if
  schedulerNOOP > 0
  filesetSize > 51200
  blockSize > 20480
  readPercentage > 0
  sequentialAccess <= 0
  then
  outcome = 8.014792 + 0.000111 filesetSize - 0.114 readPercentage
            + 0.00015 blockSize

Rule 22: [16 cases, mean 17.517969, range 7.89865 to 27.85923, est err 1.854476]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  blockSize > 16384
  readPercentage > 60
  sequentialAccess <= 0
  then
  outcome = -0.776021 + 0.000225 filesetSize + 0.00017 blockSize

Rule 23: [48 cases, mean 19.517345, range 13.3347 to 25.47197, est err 0.891073]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  blockSize > 16384
  readPercentage > 0
  readPercentage <= 60
  sequentialAccess <= 0
  then
  outcome = 3.948327 + 0.000112 filesetSize + 0.0002 blockSize
            + 0.044 readPercentage

Rule 24: [12 cases, mean 19.705690, range 13.3347 to 25.47197, est err 0.738299]

  if
  schedulerNOOP <= 0
  filesetSize > 25600
  blockSize > 16384
  readPercentage > 50
  readPercentage <= 60
  sequentialAccess <= 0
  then
  outcome = 0.312542 + 0.000186 filesetSize + 0.0002 blockSize

Rule 25: [15 cases, mean 22.816879, range 16.07982 to 29.79744, est err 1.605232]

  if
  schedulerNOOP > 0
  filesetSize > 25600
  blockSize <= 20480
  readPercentage <= 0
  sequentialAccess <= 0
  then
  outcome = 10.360359 + 0.00051 blockSize + 7.6e-05 filesetSize
```

```
Rule 26: [24 cases, mean 23.544998, range 16.40182 to 39.31592, est err 2.004764]

  if
  schedulerNOOP <= 0
  filesetSize <= 1024
  blockSize <= 16384
  sequentialAccess > 0
  then
  outcome = 23.739252 + 0.00068 blockSize - 0.18 readPercentage

Rule 27: [12 cases, mean 24.850025, range 17.15667 to 35.95248, est err 1.681448]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  blockSize <= 12288
  readPercentage <= 0
  sequentialAccess <= 0
  then
  outcome = 13.737529 + 0.000165 filesetSize

Rule 28: [9 cases, mean 25.647768, range 24.49214 to 26.89822, est err 0.450433]

  if
  schedulerNOOP > 0
  filesetSize > 25600
  blockSize > 20480
  readPercentage <= 0
  sequentialAccess <= 0
  then
  outcome = 17.892457 + 0.00023 blockSize + 1.3e-05 filesetSize

Rule 29: [20 cases, mean 26.282497, range 21.74508 to 28.52684, est err 0.668663]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  blockSize > 12288
  readPercentage <= 0
  sequentialAccess <= 0
  then
  outcome = 18.901978 + 5.7e-05 filesetSize + 0.00015 blockSize

Rule 30: [96 cases, mean 33.976761, range 26.17056 to 42.61604, est err 0.456515]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  blockSize <= 16384
  sequentialAccess > 0
  then
  outcome = 20.973984 + 0.00111 blockSize + 2e-05 filesetSize
            + 0.01 readPercentage

Rule 31: [24 cases, mean 34.191681, range 24.95808 to 54.83305, est err 2.677022]

  if
  schedulerNOOP <= 0
  filesetSize <= 1024
  blockSize > 16384
  sequentialAccess > 0
  then
  outcome = 28.417545 - 0.278 readPercentage + 0.00062 blockSize

Rule 32: [32 cases, mean 42.822277, range 26.17056 to 60.72971, est err 0.827015]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  readPercentage > 60
  sequentialAccess > 0
  then
  outcome = 19.905033 + 0.00105 blockSize + 6.1e-05 filesetSize

Rule 33: [40 cases, mean 50.671181, range 43.39243 to 58.54386, est err 0.463372]

  if
  schedulerNOOP <= 0
  filesetSize > 1024
  filesetSize <= 51200
  blockSize > 16384
  readPercentage <= 60
  sequentialAccess > 0
  then
  outcome = 21.129888 + 0.00102 blockSize + 6e-05 filesetSize

Rule 34: [48 cases, mean 52.056000, range 44.18821 to 60.72971, est err 0.483730]

  if
  schedulerNOOP <= 0
  filesetSize > 51200
  blockSize > 16384
  sequentialAccess > 0
  then
  outcome = 20.948825 + 0.00103 blockSize + 2.6e-05 filesetSize
            + 0.035 readPercentage
```

D. Glossary

benchmark is a piece of software which runs on the system under test. It produces load on the system and captures results of given metrics. 118

benchmark controller uses the benchmark drivers to coordinate the execution of benchmarks on the systems under test. 39, 118

benchmark driver is the interface between the benchmark controller and the benchmark. It launches a given benchmark, captures the output and extracts the results from the output. 39, 41, 118

benchmark harness consists of multiple components which help to plan and control the execution of benchmarks on multiple hosts, gather the results and store them for further analysis. 118

benchmark run A single execution of a benchmark on one system under test. 118

dependent variable is a variable, whose values are generated during the run of a benchmark. The values are dependent on the values of the independent variables. 5, 24, 118

experiment is a specification of one benchmark run defined by a value for each of the independent variables needed and a system under test where the benchmark run should be executed on. 41, 118

experiment series is a term used in singular form for a set of experiments on a system under test. The experiments are defined by a independent variable space. 42, 118

experiment setup defines the overall experiment configuration. It contains an experiment series for each system under test which will be involved. 118

independent variable is also called an explanatory variable or input variable. It is a variable whose values are set prior to the run of a benchmark. It includes all factors which can possibly change the outcome of a benchmark. 5, 6, 24, 41, 118

independent variable space is a set of independent variables and one or more values for each of the variables. 42, 118

system under test is the actual host or machine for which the benchmark results should be generated. 41, 118

List of Figures

List of Tables

Bibliography

[AAH+03] I. Ahmad, J. Anderson, A. Holler, R. Kambo, and V. Makhija, "An analysis of disk performance in VMware ESX Server virtual machines," in *Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on.* IEEE, 2003, pp. 65–76.

[And01] E. Anderson, "Simple table-based modeling of storage devices," HP Laboratories, SSP technical report, July 2001.

[BCR94] V. R. Basili, G. Caldiera, and H. D. Rombach, "The goal question metric approach," *Encyclopedia of Software Engineering*, pp. 528–532, 1994.

[BFS+06] F. Benevenuto, C. Fernandes, M. Santos, V. Almeida, J. Almeida, G. Janakiraman, and J. Santos, "Performance models for virtualized applications," in *Frontiers of High Performance Computing and Networking – ISPA 2006 Workshops*, ser. Lecture Notes in Computer Science, G. Min, B. Di Martino, L. Yang, M. Guo, and G. Rünger, Eds. Springer Berlin / Heidelberg, 2006, vol. 4331, pp. 427–439.

[BFSO84] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classification and regression trees*, ser. The Wadsworth and Brooks-Cole statistics-probability series. Chapman & Hall, 1984.

[BKR09] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *J. Syst. Softw.*, vol. 82, no. 1, pp. 3–22, Jan. 2009.

[CHC+01] Y. M. Chae, S. H. Ho, K. W. Cho, D. H. Lee, and S. H. Ji, "Data mining approach to policy analysis in a health insurance domain," *International Journal of Medical Informatics*, vol. 62, Issue 2, pp. 103–111, 2001.

[CKK11] G. Casale, S. Kraft, and D. Krishnamurthy, "A model of storage I/O performance interference in virtualized systems," in *31st IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2011 Workshops), 20-24 June 2011, Minneapolis, Minnesota, USA*. IEEE Computer Society, 2011, pp. 34–39.

[CW00] M. Courtois and M. Woodside, "Using regression splines for software performance analysis," in *Proceedings of the 2nd international workshop on Software and performance*, ser. WOSP '00. New York, NY, USA: ACM, 2000, pp. 105–114.

[DBC+10] B. Dufrasne, W. Bauer, B. Careaga, J. Myyrrylainen, A. Rainero, and P. Usong, *IBM System Storage DS8700: Architecture and Implementation.* IBM Corporation, International Technical Support Organization, 2010.

[DSW03] B. Dufrasne, D. Skilton, and J. Wright, *Linux with zSeries and ESS: Essentials.* IBM Corporation, International Technical Support Organization, 2003.

[Fab] Faban, "Harness and benchmark framework," last visited 2012/07/22. [Online]. Available: http://java.net/projects/faban/

[Fab11] M. Faber, "Software performance analysis using machine learning techniques," Master's thesis, Institute for Program Structures and Data Organization (IPD), Karlsruhe Institute of Technology, March 2011.

[Fri91] J. H. Friedman, "Multivariate adaptive regression splines." *Ann. Stat.*, vol. 19, no. 1, pp. 1–141, 1991.

[HBR+10] N. Huber, S. Becker, C. Rathfelder, J. Schweflinghaus, and R. H. Reussner, "Performance modeling in industry: A case study on storage virtualization," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 1–10.

[HEK05] J. Hartung, B. Elpelt, and K.-H. Klösener, *Statistik: Lehr- und Handbuch der angewandten Statistik*, 14th ed. München: Oldenbourg Wissenschaftsverlag, 2005.

[HKHR11] M. Hauck, M. Kuperberg, N. Huber, and R. Reussner, "Ginpex: Deriving performance-relevant infrastructure properties through goal-oriented experiments," in *Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS*, ser. QoSA-ISARCS '11. New York, NY, USA: ACM, 2011, pp. 53–62.

[HTF11] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, ser. Springer Series in Statistics. Springer, 2011.

[HvQHK11] N. Huber, M. von Quast, M. Hauck, and S. Kounev, "Evaluating and modeling virtualization performance overhead for cloud environments," in *1st International Conference on Cloud Computing and Services Science (CLOSER 2011), May*, 2011, pp. 7–9.

[KCK+11] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick, "IO performance prediction in consolidated virtualized environments," in *Proceedings of the second joint WOSP/SIPEW international conference on Performance engineering*, ser. ICPE '11. New York, NY, USA: ACM, 2011, pp. 295–306.

[KCK+12] ——, "Performance models of storage contention in cloud environments," *Springer Journal of Software and Systems Modeling*, pp. 1–24, 2012.

[KKB+07] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on.* IEEE, 2007, pp. 200–209.

[KLSC07] H. Kim, W.-y. Loh, Y.-s. Shih, and P. Chaudhuri, "Visualizable and interpretable regression models with good prediction power," *IIE Transactions*, vol. 39, p. 565–579, 2007.

[Kou11] S. Kounev, "Performance engineering of enterprise software systems," 2011, University Lecture, last visited 2012/06/26. [Online]. Available: http://sdqweb.ipd.kit.edu/wiki/Vorlesung_Performance_Engineering_of_Enterprise_Software_Systems_SS2011

[KRDZ10] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao, "Application performance modeling in a virtualized environment," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on.* IEEE, 2010, pp. 1–10.

[Kuh08] M. Kuhn, "Building predictive models in R using the caret package," *Journal of Statistical Software*, vol. 28, no. 5, pp. 1–26, 11 2008. [Online]. Available: http://www.jstatsoft.org/v28/i05

[Kva85] T. O. Kvalseth, "Cautionary note about R2," *The American Statistician*, vol. 39, no. 4, pp. 279–285, Nov. 1985.

[LK93] E. K. Lee and R. H. Katz, "An analytic performance model of disk arrays," *SIGMETRICS Perform. Eval. Rev.*, vol. 21, no. 1, pp. 98–109, Jun. 1993.

[MCCS08] N. Mi, G. Casale, L. Cherkasova, and E. Smirni, "Burstiness in multi-tier applications: Symptoms, causes, and new models," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, ser. Middleware '08. New York, NY, USA: Springer-Verlag New York, Inc., 2008, pp. 265–286.

[Mei08] S. Meier, *IBM Systems Virtualization: Servers, Storage, and Software.* IBM Corporation, International Technical Support Organization, 2008.

[MF02] G. Moisen and T. Frescino, "Comparing five modelling techniques for predicting forest characteristics," *Ecological Modelling*, vol. 157, no. 2, pp. 209–225, 2002.

[NKR12] Q. Noorshams, S. Kounev, and R. Reussner, "Experimental evaluation of the performance-influencing factors of virtualized storage systems," in *Proceedings of the 9th European Performance Engineering Workshop (EPEW'12), Munich, Germany, July 30-31*, 2012.

[PBH+08] L. Parziale, M. Belardi, M. Held, L. Y. Ma, L. Peckover, and K. Reed, *Linux on IBM System z: Performance Measurement and Tuning.* IBM Corporation, International Technical Support Organization, 2008.

[Qui92] J. Quinlan, "Learning with continuous classes," in *Proceedings of the 5th Australian joint Conference on Artificial Intelligence.* Singapore, 1992, pp. 343–348.

[Ste07] P. Sterlin, "Overfitting prevention with cross-validation," Master's thesis, Laboratoire d'Informatique de Paris 6 (LIP6), 2007.

[TKKO05] M. Ture, I. Kurt, A. T. Kurum, and K. Ozdamar, "Comparing classification techniques for predicting essential hypertension," *Expert Systems with Applications*, vol. 29, no. 3, pp. 583–588, 2005.

[vBK04] W. C. M. van Beers and J. P. C. Kleijnen, "Kriging interpolation in simulation: A survey," in *Proceedings of the 36th conference on Winter simulation*, ser. WSC '04. Winter Simulation Conference, 2004, pp. 113–121.

[WAA+04] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage device performance prediction with CART models," in *Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems.* Washington, DC, USA: IEEE Computer Society, 2004, pp. 588–595.

[WH11] D. Westermann and J. Happe, "Performance cockpit: Systematic measurements and analyses," in *Proceedings of the second joint WOSP/SIPEW international conference on Performance engineering*, ser. ICPE '11. New York, NY, USA: ACM, 2011, pp. 421–422.

[WHHH10] D. Westermann, J. Happe, M. Hauck, and C. Heupel, "The performance cockpit approach: A framework for systematic performance evaluations," in *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*. IEEE, 2010, pp. 31–38.

[WHKF12] D. Westermann, J. Happe, R. Krebs, and R. Farahbod, "Automated inference of goal-oriented performance prediction functions," in *27th IEEE/ACM International Conference On Automated Software Engineering (ASE 2012)*, September 2012.