School of Electronics, Electrical Engineering & Computer Science

Performance Evaluation of Shared Data Access in Enterprise Systems

Stephan Kraft, M.Sc.

A Thesis Submitted To Queen's University Belfast In Fulfillment Of The Requirements For The Degree Of Doctor Of Philosophy

November 20, 2012

NOTHING IN THE WORLD CAN TAKE THE PLACE OF PERSISTENCE.

TALENT WILL NOT; NOTHING IS MORE COMMON THAN UNSUCCESSFUL MEN WITH TALENT.

GENIUS WILL NOT; UNREWARDED GENIUS IS ALMOST A PROVERB.

EDUCATION WILL NOT; THE WORLD IS FULL OF EDUCATED DERELICTS.

PERSISTENCE AND DETERMINATION ALONE ARE OMNIPOTENT.

THE SLOGAN "PRESS ON" HAS SOLVED AND ALWAYS WILL SOLVE THE PROBLEMS OF THE HUMAN RACE.

CALVIN COOLIDGE (1872 - 1933)

Abstract

Cloud computing and virtualization technologies have created a shift in data center usage patterns, where the sharing of resources and on-demand service offerings can result in varying workload mixes and sudden bursts in application requests. Enterprises may benefit from cloud computing by consolidating multiple Virtual Machines (VMs) on the same virtualized server and sharing CPU, memory and storage devices. However, the shared access to hardware and software resources can lead to significant performance degradation due to contention and workload interference effects. A decisive factor of enterprise application performance are the latencies of data access operations, e.g., read requests to data sets stored in a database. In-memory database technologies try to omit the time-consuming access to disk storage and promise performance gains by maintaining large data sets in random access memory.

The first part of this thesis introduces a set of performance models for evaluation of consolidated disk workloads which are typical of virtualized data centers. In particular we consider several VMs managed by a single Virtual Machine Monitor (VMM) submitting disk I/O intensive workloads to a shared storage device. Our methodology requires first a study of VMs when running in isolation, i.e. with only a single VM running. Based on collected measurements we propose models for homogeneous and heterogeneous workload mixes to forecast the impact of consolidation on I/O performance.

In the second part we propose and evaluate admission control policies for in-memory database queries. Our methodologies use system measurements and aim to prioritize mixes of database queries consuming CPU resources efficiently. The performance of database queries can be significantly impacted by the interplay between queries running at the same time. The interplay effects are not limited to heightened resource contention at the CPU cores, but may also involve delays due to memory access or due to the internal characteristics of the software such as limited application threads. We demonstrate by trace-driven simulation that our policies are more efficient than popular policies such as first-come first-served and shortest-expected-processing-time first.

Content

List of Figures

List of Tables

Acknowledgements

I need to thank my employer SAP, the InvestNI projects MORE and VIRTEX, and the EU FP7 project CumuloNimbo for the generous funding of my work. This thesis would not have been possible without the everyday support of my colleagues at SAP Research in Belfast. In particular, I am in debt to the performance team for granting me the perfect combination of freedom, direction, and patience: Stephen Dawson, Giuliano Casale, Sergio Pacheco-Sanchez, and Tariq Ellahi. Thanks to our lab director Ben Greene for his continuous support. I am grateful for the help and assistance of Diwakar Krishnamurthy and Alin Jula.

My supervisors from Queen's University Belfast, Des Greer and Peter Kilpatrick, have been invaluable guides throughout the past three years. In addition, I need to express my sincere gratitude for receiving an university travel grant and the Emily Sarah Montgomery Scholarship.

My deepest respect and gratefulness goes to my families who taught me that hard work and persistence pays off in the end. Lilo, Horst, and Danny not once lost their belief in my abilities. Siobhán has been a true inspiration and a lifesaver especially in the times when work did not go as planned. Thanks to Kris, Chuck, and Dane for making me part of their lives and their "*kind ignorance and ability to treat me like a piece of furniture*[Van12]".

[Van12] VANDENBERG, C. Internal communication (2012).

Abbreviations

BI	Business Intelligence
CART	Classification and Regression Trees
CCDF	Complementary Cumulative Distribution Function
CDF	Cumulative Distribution Function
CFQ	Completely Fair Queueing
CTMC	Continuous Time Markov Chain
CV	Coefficient of Variation
EQMS	External Queue Management System
FCFS	First-Come-First-Served
FCR	Finite Capacity Region
FFSB	Flexible File System Benchmark
FJQN	Fork-Join Queueing Networks
FS	Fair-Share
GA	General Availability
GPS	Generalized Processor Sharing
HDD	Hard Disk Drive
JMCQ	Join-Minimum-Cost-Queue
JSQ	Join-The-Shortest-Queue
KVM	Kernel-based Virtual Machine
LAN	Logical Area Network
LBA	Logical Block Address
LUN	Logical Unit Number
MAP	Markovian Arrival Process
MPL	Multi-Programming Limit
MQO	Multi Query Optimization
MVA	Mean Value Analysis
NAS	Network Attached Storage
NTP	Network Time Protocol
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PDF	Probability Density Function
PM	Postmark
QUB	Queens University Belfast
RAID	Redundant Array of Inexpensive Disks
RAM	Random Access Memory
SAN	Storage Attached Network

SCSI	Small Computer System Interface
SEPT	Shortest Expected Processing Time First
SFQ	Start-Time Fair Queueing
TPC	Transaction Processing Council
VMFS	Virtual Machine File System
VMM	Virtual Machine Monitor

1 Introduction

The performance of data access operations can play a fundamental role in enterprise IT systems. Large latencies of read requests to data sets stored in a database, e.g., may lead to discontinued customer sales transactions. An important factor influencing the data access performance can be the sharing of storage resources by multiple consumers. The sharing of CPU, memory and storage devices is enabled by cloud computing and virtualization technologies which facilitate the consolidation of multiple Virtual Machines (VMs) on the same virtualized server. Virtualization offers to enterprises the benefits of more efficient resource utilization and flexible workload placement [139, 186]. However, the shared access to hardware and software resources can lead to significant performance degradation due to contention and interference between different consolidated workload types [124].

In shared environments the performance evaluation of data access operations is complicated by varying workload mixes and sudden bursts in application requests. Enterprises commonly rely on disks as a medium for non-volatile data storage. The first part of this thesis is motivated by the need for predictive models to forecast the performance implications of disk workload consolidation in virtualized environments. Our models consider a system where multiple VMs are installed on the same virtualized server with shared disk storage. In essence we aim to predict the performance of disk requests as perceived by the VM's operating system when running at the same time as other VMs, i.e. in consolidation. The prediction is based on measurements collected inside the VM when it is running by itself, i.e. in isolation. Our models can assist data centre administrators in making VM placement decisions, e.g., for *VM Appliances*: Preconfigured VMs entailing software applications such as web and mail servers. The methodology requires the collection of system traces of VMs running in isolated environments. Using the traces from the trace repository system administrators can accurately predict the performance of disk requests when placing VMs in consolidation on a single server.

In the second part of the thesis we progress to a performance evaluation of systems maintaining data sets in random access memory (RAM). Recently RAM has become a competitive and economically viable storage option for volatile data. Compared to the time-consuming data access operations to disks, the access times to data residing in system memory are considerably lower. In-memory database technologies try to leverage this advantage and promise performance gains by maintaining large data sets in RAM. In particular, we study the performance of a number of database queries running at the same time in an in-memory database system. We propose and evaluate a number of policies governing the order of query admission to the database by prioritizing certain query types at the expense of others. Our policies may be employed as part of an admission controller external to the database system aiming to prioritize query mixes delivering good performance. The external admission controller may

intercept incoming client queries for change of the admission order to the in-memory database application according to the admission policies.

The quantitative evaluation of computer systems has been an important research area for a long time [130, 191]. Performance can be a crucial factor during all stages of the system life cycle and therefore needs to be taken into account during design, development, configuration, and operation [108]. This relevance is reflected in a large number of models and formalisms that have been proposed over the years to capture and predict the quantitative behavior of computing systems.

Software performance models aim to integrate performance prediction into the early stages of the software development process [47]. Typically approaches require the availability of software artifacts, e.g. design documents, that are subsequently enriched with non-functional information. Instead, our work focuses on *system performance models* and involves the study of a system implementation. Throughout the thesis the term *performance model* refers to models of the latter type.

System performance modeling research has traditionally been based on concepts from queueing theory. Computer systems generally consist of a number of hardware and software resources. A set of jobs compete for limited resource access most likely resulting in queues and delays. It is therefore natural to model the system by a network of interconnected queues [108]. Modern approaches for performance evaluation also include, e.g., Generalized Stochastic Petri Nets [138], Stochastic Process Algebra [110], and Stochastic Automata Networks [150].

The objective of performance models is essentially the prediction of metrics such as resource utilization, response times, throughputs, queue lengths, and queueing delays. Models may be distinguished according to the way these metrics are derived. *Analytical models* construct a mathematical representation of the system, the main difficulty being that the domain of tractable models is rather limited [121]. *Simulation models* have the advantage of being more general and flexible. However, modeling the system under study in full detail may result in simulations with large computational costs.

In this thesis we rely on concepts from queueing theory and mainly use simulation to model the systems under study. In addition to simulation, we apply a number of simple linear equations for models of VM consolidation scenarios. Our work follows a system approach in that it makes use of detailed system trace measurements thereby treating the virtualized server and the in-memory database as black boxes. Maintaining a black box view can be advantageous since such detailed information may not always be available and usually complicates the model.

1.1 Research Approach

The first part of this thesis presents a set of performance models for evaluation of consolidated disk workloads which are typical of virtualized data centres. The performance of I/O-bound applications is dominated by the time required by the operating system to schedule read and write operations and by the response times of the storage devices in completing such requests. Since changes in the workload, as well as in the software and hardware environments, can affect the latency of disk I/O requests, it is often useful to define performance models to anticipate the effects of a change.

In particular we consider several VMs managed by a single Virtual Machine Monitor (VMM) submitting disk I/O intensive workloads to a shared storage device. Our methodology requires first the study of VMs when they run in isolation, i.e. with only a single VM running. Based on collected measurements we propose two types of models to forecast the impact of consolidation on I/O performance. Specifically, for each VM of interest we collect traces of arrival times, estimated service times, and arrival queue lengths for I/O requests. We develop two modeling techniques that embody this approach. We start by developing what we denote as the Homogeneous model to handle scenarios where multiple VMs running the same type of workload are consolidated. We then extend this technique by developing what we denote as the Decomposition model. In addition to handling scenarios where consolidated VMs run heterogeneous workloads, this technique supports the distinction between read and write requests.

We first consider the Homogeneous model. This model is based on a trace-driven simulation that uses start-time fair queueing (SFQ), a popular implementation of fair share scheduling which is adopted in VMMs. Motivated by the fact that a trace-driven simulation in such a black box view of the system typically fails to accurately predict I/O request response times under consolidation, we define an iterative algorithm for optimal parameterization of the simulation model. Specifically, the algorithm estimates the performance impact of VMM level I/O optimizations such as the splitting of requests and offers a search technique for model calibration.

The Decomposition model approach distinguishes between read and write requests and models each request type separately. In addition to mean I/O request response times the linear predictor formulas also forecast throughputs and read/write request mixes in consolidation. Finally, this approach is complemented with a simulation model to predict latency distributions. The main contributions of the first part of the thesis are threefold.

1. Our methodology allows us to parameterize models based on system traces obtained from inside VMs in isolation experiments, i.e. with only a single VM running on the server. It requires minimal information from the VMM thereby effectively treating the VMM as a black box. In addition this obviates the requirement to collect model training data for different VM consolidation scenarios.

2. We develop a trace-driven simulation model for the prediction of disk request latencies with several VMs in consolidation, i.e. running concurrently on the same physical server. The queueing model is enhanced with an iterative calibration technique that approximates parameterization inaccuracies in the absence of detailed VMM level measurements.

3. A measurement-based model defines a set of simple analytical estimators to approximate consolidation performance of read/write disk requests separately. This approach is complemented with a simulation model for prediction of disk request latency distributions.

In the second part of the thesis we progress to an investigation of how the transition to in-memory technologies affects the performance of consolidated database query workloads. The small data access latencies of in-memory databases in combination with parallelization capabilities of multicore servers enable the near real-time execution of often computationally-intensive business intelligence applications on very large data sets.

In particular we propose and evaluate four admission control policies for in-memory database queries. Our methodologies use system measurements and aim to prioritize mixes of database queries consuming CPU resources efficiently. The performance of database queries can be significantly impacted by the interplay between queries running at the same time. The interplay effects are not limited to heightened resource contention at the CPU cores, but may also involve delays due to memory access or due to the internal characteristics of the software such as limited application threads. In summary the second part of the thesis makes four contributions:

1. We propose admission control policies derived from (i) mean job queue lengths, (ii) CPU utilization, and (iii) a no-interference baseline measure for query class pairs computed from ratios of queue lengths and utilization. The policies ignore I/O subsystems and therefore are highly applicable to in-memory databases. In essence, our policies maintain a table that characterizes the interplay and prioritizes query classes that are able to use resources more efficiently, typically at the expense of other query classes.

2. For evaluation of admission control policies we develop a trace-driven simulation model. The closed queueing model is validated against system measurements.

3. Using the trace-driven simulation model and simplified synthetic workloads we present a comparative evaluation of the proposed admission control policies. In particular, we compare throughput, mean response time, and mean weighted slowdown results to the established policies first-come first-served (FCFS) and shortest-expected-processing-time-first (SEPT) [149].

4. We further validate the queue length and no-interference baseline policies using TPC-H [12] type workloads and system traces from a commercial solution, SAP HANA [86], a product aimed at improving the performance of complex business intelligence workloads using

in-memory database technology. The SAP HANA database is a memory-centric data management system leveraging the capabilities of modern main memory and multi-core CPU hardware.

Summarizing, in this thesis we evaluate the performance of data operations in environments with shared resource access. The thesis is structured into two parts. In the first part we present predictive models for disk I/O request performance with consolidated storage workloads. The second part studies interplay effects between database queries and introduces admission control policies for in-memory database applications.

1.2 Project Organization

This section explains the roles of the parties involved in the PhD project. The thesis has been completed during my time of employment with *SAP* [10] and enrollment at *Queens University Belfast (QUB)*. The following people have been involved in the research project:

- Dr. Des Greer holds a Senior Lecturer position at QUB and is the first thesis supervisor
- Dr. Peter Kilpatrick holds a Senior Lecturer position at QUB and is the secondary thesis supervisor
- Dr. Stephen Dawson is a SAP colleague, the manager of the project, and third thesis supervisor
- Dr. Giuliano Casale is a Lecturer at *Imperial College London*. He has been a full time colleague at SAP during the first year of the project and following his transfer to Imperial College acted as an external consultant and mentor.
- Dr. Diwakar Krishnamurthy holds a position as Associate Professor at the *University of Calgary*. He advised the project during a six months sabbatical at SAP. After his return to Calgary the collaboration continued resulting in joint publications.
- Dr. Alin Jula and Dr. Sergio Pacheco-Sanchez are both SAP colleagues and have been involved in the project as publication co-authors.

The contributions presented in the thesis are entirely the work of the author. Where the work of others in the project has been used or built on, this is specifically stated in the thesis.

1.3 Thesis Structure

The remainder of the thesis is organized as follows.

- In **Chapter 2** we introduce some key concepts and notation that are essential to the understanding of the following chapters.
- **Part I** of the thesis consists of Chapters 3 and 4.
- The work presented in **Chapter 3** introduces a trace-driven simulation model for prediction of disk I/O request latencies in homogeneous workload consolidations. The methodology includes an iterative algorithm for the optimal parameterization of the simulation model.
- In **Chapter 4** a measurement-based decomposition approach is proposed that distinguishes between read and write requests and models each request type separately. The linear predictor formulas forecast average disk request latencies, throughputs, and read/write request mixes for heterogeneous workload consolidations. The decomposition approach is further embodied in a simulation model for prediction of disk request latency distributions.
- **Part II** of the thesis consists of Chapter 5.
- Novel admission control policies for in-memory database queries are proposed and evaluated in **Chapter 5**. The policies are derived from system measurements and evaluated using trace-driven simulation. We first present a comparative study of the policies using simplified synthetic workloads. Then we proceed to more realistic workloads based on the TPC-H benchmark.
- In **Chapter 6** we draw final conclusions and outline future research directions.

2 Background

This chapter introduces the general concepts and terminology used throughout the thesis. Section 2.1 explains the tools and techniques employed for workload characterization and presents the most prevalent characteristics of disk I/O workloads. An introduction to queueing networks is provided in Section 2.2. We then progress from theoretical formalisms to system related topics. Section 2.3 establishes the thesis context of virtualization and Section 2.4 focuses on storage systems. We complete the chapter with an introduction to in-memory databases in Section 2.5 and a brief summary in Section 2.6.

2.1 Workload Characterization

The quantitative evaluation of computer systems involves the collection of monitoring measurements while the system is being subjected to a particular *workload* [113]. A workload consists of events generated by the environment in which the system is used, e.g., the arrival times and service requirements of incoming client requests. The characterization of workloads [63] is one of the central issues in performance evaluation since it is not always obvious what aspects of the workload are important, in how much detail the workload should be recorded, and how the workload should be represented and used [121]. In this section we provide a high-level introduction of tools and techniques for workload characterization and their application to disk I/O workloads.

2.1.1 Benchmarking Tools

Benchmarks are a commonly adopted tool to generate predefined and reproducible workloads in order to study a system under controllable conditions. The benchmarking techniques for storage systems can be classified into three categories [190]:

- *Macrobenchmarks.* Performance is tested against a particular workload that is meant to emulate some real-world workload. These types of benchmarks may also be referred to as *synthetic benchmarks* and usually comprise multiple file system operations. While Macrobenchmarks are a valuable tool for the evaluation of overall system performance the creation of realistic workloads is challenging [90].
- *Trace Replays.* System traces of data access operations are recorded on a real-world installation and later replayed using a software tool [115, 210]. The advantage of using traces is that the key characteristics of the workload are realistically captured and preserved. Traces can be collected from measurements or obtained in third party repositories [15, 27].

- *Microbenchmarks.* These benchmarks comprise only a few file system operations to isolate their specific overheads within the system. Such workloads can be useful to test the performance of specific system parts or study worst-case behavior.

There is a large number of storage benchmarks available for scientific studies [190]. In this thesis we aim to evaluate our proposed methodologies based on a range of workloads with distinct characteristics. With the goal of using workloads with properties close to real-world applications we choose the macrobenchmarks *Postmark (PM)* [9], *Flexible File System Benchmark (FFSB)* [3], *Filebench* [26], and *TPC-H* [12]. See Appendix D for the detailed benchmark configurations.

PM. This benchmark aims to emulate typical disk I/O transactions of electronic mail, netnews, and e-commerce applications [119]. The workload consists of a large number of short-lived files of small sizes executed by a single thread and comprises a mix of data-intensive and metadata-intensive operations. A benchmark run starts with the creation of a pool of text files in uniformly distributed sizes where upper and lower file size bounds are configurable. Once all files have been created PM performs create or delete operations paired with read or append operations. The total numbers of files, sub-directories, file-size ranges, and performed operations, as well as reads-to-append and creates-to-deletes ratios can be configured. PM has been widely deployed in the research community [42, 58, 123, 127, 135, 153, 167, 185, 202]. For the work in this thesis we use PM in version 1.51-7 [9].

FFSB. This workload generator may be used to emulate large file operations inherent in web, file, and mail servers. FFSB workloads are highly configurable and allow the specification of profiles that are passed as input to the multi-threaded benchmark application. At the beginning of a benchmark run a number of directories and files are created that are then subsequently used to perform seven basic file operations, e.g., read, write, create, delete, and append. Multiple groups of threads with distinct operation mixes may be configured. An interesting feature of FFSB is the support of sequential and random access operations. In this thesis the FFSB version 6.0-rc2 is deployed [3]. FFSB has also been used in related work [58, 179].

Filebench. A large variety of emulated application and microbenchmark workloads can be generated with Filebench [26]. The tool features a number of predefined workload types, e.g., 'web server', 'mail server', and 'file server', as well as a scripting language for definition of custom workloads. This flexibility may explain the popularity of filebench in recent scientific studies [34,85,102,104,105,181,189]. Workload definitions include the directory structure, i.e., average file size, directory depth, and the total number of files. In addition, the tool supports configuration of alpha parameters governing file and directory sizes that are based on a gamma distribution. Multiple threads can be configured to execute groups of file system operations, e.g., read, write, and append. In this thesis we use predefined emulated application workloads shipped with Filebench version 1.4.8.fsl.0.7.

TPC-H. The *Transaction Processing Performance Council (TPC)* publishes a number of standardized transaction processing and database benchmarks for quantification of system performance in industry. TPC-H is a decision support benchmark consisting of 22 business oriented database queries and concurrent data modifications [12]. The benchmark is mainly used for evaluation of database performance [122, 142, 211], but can also be found in work regarding file system and storage research [93, 167, 198].

2.1.2 Workload Characterization Techniques

Workload characterization involves the analysis of monitoring data for estimation of the *workload parameters*. For example, clustering techniques [148, 166] or Independent Component Analysis [183] may be employed to determine request classes from measured system logs. In this thesis we assume types and classes of requests to be known a priori and analyze data samples to characterize statistical properties of parameters, e.g., for request interarrival, service, and response times. Two key concepts in statistical analysis are the independence of events and the randomness of variables. Events are called independent if the observation of one event does not in any way affect the probability of occurrence of another event. A random variable receives a value from a set of values with a specified probability.

The most compact form for characterization of workload parameters is a single number that summarizes the data set of n observed samples. A commonly used statistic is the sample *arithmetic mean*, denoted $\overline{X}$. An alternative is the *median* which is obtained by sorting the samples in an increasing order and taking the x_i value in the middle of the sorted series. Since the variability of workload parameters has been identified to have large impacts on system performance [147, 161], it is often useful to specify the dispersion of sample data, e.g., by the *variance*:

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{X})^2. \tag{2.1}$$

The *standard deviation*, denoted σ, is defined as the square root of the variance. Interpretation of σ may be easier as it is expressed in the same unit as the mean. Additionally, the *coefficient of variation (CV)* may be useful for comparisons as it is normalized by the sample mean:

$$CV = \frac{\sigma}{\overline{X}}. \tag{2.2}$$

The events in a time series do not always occur independently of each other. For example, disk schedulers aim to organize data access operations to contiguous disk sectors sequentially. Thus,

the subsequent *logical block addresses (LBAs)* in a disk trace may be correlated depending on spatial locality. The *autocorrelation* is a metric to measure the similarity between events as a function of the time separation, i.e. the time-lag, between them. Autocorrelation is defined as

$$R(\tau) = \frac{(x_t - \overline{X})(x_{t+\tau} - \overline{X})}{\sigma^2}, \tag{2.3}$$

where x_t and $x_{t+\tau}$ are events at times t and $t + \tau$, respectively. A value of $R(\tau) = 1$ indicates perfect correlation.

Another way of summarizing a data set is to present its statistical distribution. A continuous random variable X can assume all values in the interval $[a, b]$, where $-\infty \leq a \leq b \leq \infty$, and is described by its *cumulative distribution function (CDF)*. The CDF specifies the probability that the random variable X takes values less than or equal to x, for every x [56]:

$$F_X(x) = P(X \leq x). \tag{2.4}$$

The derivative of the CDF $F_X(x)$ is called the *probability density function (PDF)* of x:

$$f_X(x) = \frac{dF_X(x)}{dx}. \tag{2.5}$$

One of the most important distributions in performance modeling is the *exponential distribution*, since it can often be used to approximate request interarrival and service times [56]. The CDF of an exponentially distributed random variable X is given by:

$$F_X(x;\lambda) = \begin{cases} 1 - e^{-\lambda x} & 0 \leq x \leq \infty \\ 0 & \text{otherwise,} \end{cases} \tag{2.6}$$

where λ denotes the parameter of the random variable. For an exponentially distributed random variable with parameter λ the following relations hold:

$$\begin{aligned} \text{PDF} \quad & f_X(x) = \lambda e^{-\lambda x}, \\ \text{mean} \quad & \overline{X} = \frac{1}{\lambda}, \\ \text{variance} \quad & \sigma_X^2 = \frac{1}{\lambda^2}, \\ \text{coefficient of variation} \quad & \text{CV}_X = 1. \end{aligned}$$

Hence, the mean value $\overline{X}$ completely determines the exponential distribution. While the expo-

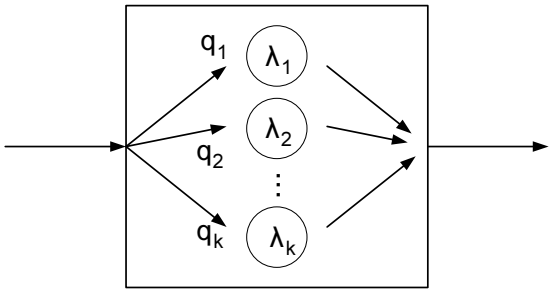


Figure 2.1: Example Model of Hyper-Exponential Distribution.

nential distribution is easy to compute and has many useful properties with analytic tractability, more complex distributions need to be applied for approximation of data sets with high variability. In cases of empirical distributions with CV > 1 the *hyper-exponential distribution* may be considered. Figure 2.1 shows an illustration of a model following a hyper-exponential distribution where k phases with exponentially distributed values and parameters λ_i are arranged in parallel. The probability that a value is given based on the jth phase is q_j, where $\sum_{j=1}^{k} q_j = 1$. The CDF of a hyper-exponentially distributed random variable X is given by:

$$F_X(x) = \sum_{j=1}^{k} q_j(1 - e^{-\mu_j x}), \quad x \geq 0. \tag{2.7}$$

In addition, the following relations hold:

$$\begin{aligned}
\text{PDF} \quad & f_X(x) = \sum_{j=1}^{k} q_j \mu_j e^{-\mu_j x}, \quad x > 0, \\
\text{mean} \quad & \overline{X} = \sum_{j=1}^{k} \frac{q_j}{\mu_j} = \frac{1}{\mu}, \quad x > 0, \\
\text{variance} \quad & \sigma_X^2 = 2 \sum_{j=1}^{k} \frac{q_j}{\mu_j^2} - \frac{1}{\mu^2}, \\
\text{coefficient of variation} \quad & \mathrm{CV}_X = \sqrt{2\mu^2 \sum_{j=1}^{k} \frac{q_j}{\mu_j^2} - 1} \quad \geq 1.
\end{aligned}$$

The parameters μ_1, μ_2 of a hyperexponential distribution with $k = 2$ phases can be readily approximated with a given sample mean $\overline{X}$ and CV_X [56].

Statistical distributions provide means to describe static properties of workload parameters. In cases were parameter values evolve dynamically over time a realistic approximation additionally needs to capture the order of events in the description. *Markov processes* [129] provide a powerful tool to analyze the dynamic time-varying behavior of workload parameters and sup-

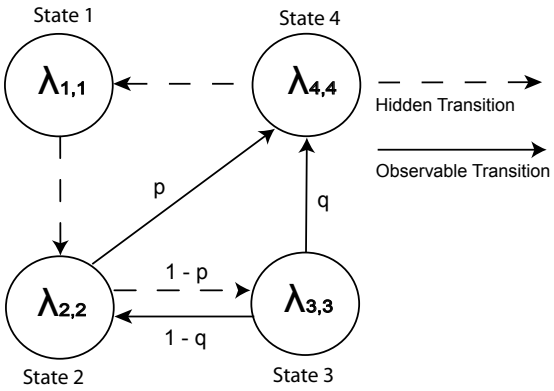


Figure 2.2: Example Markovian Arrival Process (MAP) [160].

ply the fundamental theory of queueing networks, described in Section 2.2. A Markov process with discrete state space can be described by a *Continuous Time Markov Chain (CTMC)* [56] consisting of a finite number of states and transition rates between those states. *Markovian Arrival Processes (MAPs)* [155] are a compact, tractable and expressive class of stochastic models that generalize the CTMC framework to capture not only the moments in a probability distribution, e.g. mean or variance, but also the autocorrelation in a time series. In this thesis we employ MAPs as a compact mathematical model of a series of service times. Specifically, we collect measurement samples of service times, use available tools from the *KPC-Toolbox* [69] to fit the measured service times to a MAP, and subsequently deploy the MAP to generate service times with identical statistical properties as the measured ones.

In what follows we use an introduction from related work [160] to explain the basic concepts of MAPs. Figure 2.2 illustrates an example MAP composed of $J = 4$ states. The active state at time t is $X(t) \in \{1, 2, \ldots, J\}$. Assuming the model is in state k, it spends t_k time before moving into state $j \neq k$; we assume that t_k follows an exponential distribution $\Pr(t_k = t) = \lambda_{k,k} e^{-\lambda_{k,k} t}$, thus $X(t)$ is a CTMC. The destination state j after a move is selected according to probabilities $p_{k,j}$, $\sum_{j=1}^{J} p_{k,j} = 1$.

A MAP extends a CTMC by introducing the semantics for defining interarrival times. Upon moving from state k to j a MAP defines probabilities $p^h_{k,j}$ and $p^o_{k,j}$, $p^h_{k,j} + p^o_{k,j} = p_{k,j}$, that the state transition will be *hidden* or *observable*, respectively. The only effect of hidden transitions is a change of the active state $X(t)$. Observable transitions additionally lead to the emission of a *sample*. Relating this property to our specific application of MAPs, it is evident that a service time sample of a measured trace is modeled in a MAP as the time elapsed between successive activation of any two observable transitions, i.e., the interarrival times at observable transitions.

For example, if the MAP depicted in Figure 2.2 is initialized in state 1 where it remains for time t_1, moves to state 2 spending t_2 units of time, and then takes the transition to state 4, then the sample value s_0 is generated by summation of the sojourn times $s_0 = t_1 + t_2$. The next

sample s_1 is similarly generated starting from state 4; thus, the time spent in state 4 is included in s_1. Successive visits to the same state are all cumulatively accounted for in the generated sample value. Note also that since sample s_i is generated according to the target state of the observable transition that defines s_{i-1}, a careful definition of observable state transitions can create statistical correlations between consecutive samples generated by a MAP.

Following these definitions, a MAP can be used to generate arbitrary numbers of samples s_i, $i \geq 0$. If the MAP parameters are appropriately chosen, the statistical properties of these samples can be imposed in order to fit the characteristics of a measured trace. Once this is achieved, the MAP provides a mathematical model for the trace. For example, if s_i is interpreted as the interarrival times between successive I/O requests to a disk, then the MAP is a model for the arrival process to the storage device. In this thesis a MAP is fitted to model the service times of disk I/O requests and then used to generate s_i service time samples with similar statistical properties as the measured trace.

2.1.3 Workload Characterization of Disk I/O Workloads

This section highlights performance relevant properties of disk workloads that should be considered during workload characterization. In particular, we describe disk I/O request types, arrival patterns, and service times.

Request type. From a system performance perspective storage access operations may be categorized into synchronous and asynchronous request types [171]. For example, application threads submitting a read request often block until the result has been fetched from the storage device [179]. Conversely, write requests are often completed asynchronously once they are written to system memory [21, 99]. A fine-grained taxonomy of I/O request types is presented in [91]: *Time-critical* requests cause the submitting process to halt execution until the request is completed; *time-limited* requests must be completed within a certain time limit otherwise they become time-critical; requests that need to complete for maintenance of a stable system state, but do not impose blocking of the submitting process, are classified *time-noncritical*. Time-critical and time-limited requests directly affect the performance of the submitting process, while time-noncritical requests affect performance indirectly, e.g., by *interfering* with the completion of time-critical requests.

An extensive measurement study on UNIX systems has shown that most real workloads consist of a mixture of synchronous and asynchronous requests [171]. The different nature of request types can lead to varying feedback effects that complicate workload characterization and modeling. For example, the time taken to complete a synchronous read request might affect the submission time of a subsequent request. Feedback effects in storage workloads have been identified to have significant impact on performance [90]. Workload characterization may

address feedback effects with *open* and *closed* models [92]. An open model assumes there are no feedback effects between individual request completions and subsequent arrivals. Closed models correlate the arrival times of requests with the completion times of the previous ones (see Section 2.2 for a more complete explanation of open and closed models).

Arrival pattern. A disk request may be characterized by its device ID, read/write type, start block address of the access location, request size, and arrival time. The sequence of the first four parameters is known as the *access pattern*, while the sequence of time instants in which access operations are submitted to the storage device is commonly referred to as *arrival pattern* [90]. Disk arrival patterns have been shown typically to exhibit high burstiness [96, 101, 169, 171]. An arrival pattern is bursty if it contains interspersed periods of small and large interarrival times. A burst comprises a large number of requests condensed in a small time interval, e.g., related work determines bursts in a time series by including all consecutive I/O requests whose interarrival times fall below a predefined threshold value [171]. Workloads with high burstiness exhibit longer and more intense periods of high load relative to the periods with low load [90].

Burstiness has been identified as an important source of performance degradation [67, 147]. Thus, realistic models of interarrival times need to consider the correlation in bursty arrival patterns. In [97] the authors analyze disk traces for burstiness and present a stochastic model with ON/OFF periods. A statistical model capturing access and arrival patterns of disk traffic is developed in [199]. The approach in [200] uses machine learning techniques to build *Classification and Regression Trees (CART)* from disk traces for modeling burstiness in access and arrival patterns.

Service time. The service requirements of disk requests are a function of multiple factors, e.g., spatial locality, caching along the I/O path, request size, and properties of the storage device technology. Access patterns can be either sequential or random. Sequential requests access a sequence of contiguous block addresses on the disk device, while random request accesses may be dispersed across the block device address space. The service times of sequential and random requests can significantly differ [99], since sequential requests are most directly affected by the transfer speed of the disk drive, while random requests are most directly affected by disk seek time [21, 88]. For example, if a single motion of a disk arm due to mechanical constraints takes the static time t, then moving the disk arm to serve n requests produces lower per request service times compared to serving $n - 1$ requests.

Reducing disk seek times can substantially improve performance [188]. Disk schedulers aim to maintain sequential access patterns by using *elevator algorithms* that reorder queued requests in increasing block address numbers and possibly merge requests with contiguous block addresses into larger units. In addition, sequential read requests can also take advantage of read-ahead caching, which assumes that an application reading from disk block n will next

ask to read from the subsequent disk blocks. The read-ahead mechanism decreases seek times by caching the following disk blocks $n+1$, $n+2$, etc., into memory, but is turned off once the system detects a non-sequential file access [21].

There are a number of approaches that aim to approximate the service times of disk I/O requests with detailed models of device internals. In [184, 196] the authors develop an analytical model for disk drives with read-aheads and request reordering. The service time computation considers, e.g., seek time and request size. The authors in [200] present a CART model maintaining a black box view of the storage device using disk traces with spatial locality information.

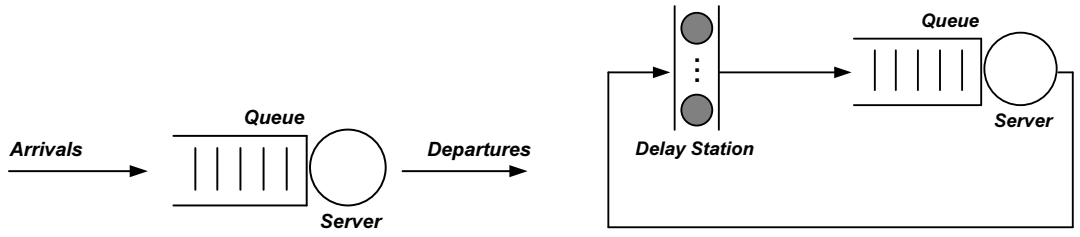
In this thesis we evaluate closed and open models to capture the effects of the mostly synchronous and asynchronous read and write requests, respectively. For accurate handling of burstiness in interarrival times and spatial locality in service times we use detailed system traces as model parameters. The traces store access and arrival patterns as measured in the reference system and therefore include bursts in arrivals, as well as service times affected by elevator algorithms and caches. Measured traces are replayed in trace-driven simulations.

2.2 Queueing Networks

Queueing networks [56, 130] are a powerful and versatile tool used to model and analyze environments with shared resources, e.g, production, communication, and computer systems. A queueing network model of a computer system consists of a collection of *service stations*, i.e. *servers*, representing the system resources. The servers are connected via directed paths and provide service to a collection of *customers*, e.g., representing jobs, users, requests, or transactions. Customers move between the servers and competition for resource service corresponds to queueing into the servers [46, 121].

Queueing networks can be classified into *open* and *closed* models. In an open model customers arrive from an external source and depart after receiving some service. The number of customers in the system may vary over time. Figure 2.3 (a) shows an example of an open model with external arrivals, departures, a single server, and a queue. In closed models the number of customers in the system stays constant and is denoted as *population*. Figure 2.3 (b) depicts an example of a closed model where customers circle to a server and after each round of service enter a *delay station* where they remain for a *think time* interval before taking another trip to the server. *Single-class* models consist exclusively of statistically identical customers, while *multi-class* models are used to represent workloads with multiple distinct customer types.

A queueing network model can be evaluated with analytical methods or using simulation to derive some performance measures, e.g., resource utilization, throughput, and customer re-



(a) Open Model (b) Closed Model

Figure 2.3: Example Open and Closed Queueing Network Models.

sponse time. The stochastic process underlying a queueing network is typically a discrete space continuous time Markov process where the state definition includes the number of customers in each queue. Consider a network with M servers and let n_i denote the customer population at node i, $n = (n_1, ..., n_M)$ the network joint queue length, and $\pi(n) = \pi(n_1, ..., n_M)$ the stationary joint queue length distribution. In cases where the queueing network is stable the stationary state probability vector of the Markov process, denoted π, is defined by the linear system:

$$\pi \quad Q = 0, \tag{2.8}$$

where Q is the transition rate matrix [46]. Stochastic analysis of queueing networks derives the performance indices from the stationary state distribution of the process. While the generality of this approach is appealing, the linear system might not always be tractable due its computational complexity: The state space may grow exponentially depending on the number of servers, customers, and customer classes in the network.

Product form queueing networks [49] satisfy a number of constraints allowing the definition of efficient analytical algorithms to evaluate their performance [113]:

1. The servers use one of the following *service disciplines*: First-come-first-served (FCFS), processor sharing (PS), infinite servers (ISs or delay station), or last-come-first-served preemptive-resume (LCFS-PR).

2. Customers can belong to multiple *classes*. Each class is characterized by a different set of routing probabilities and service time distributions at the resources. Service time distributions can be load-dependent, but they can depend only on the load at the station the served job is currently at.

3. If a resource has FCFS scheduling, to have a product-form it is additionally requested that the mean *service times* of all classes are identical. Since each class can visit a resource a different number of times, due to the different routing probabilities, the total amount of

service requested at a resource can still differ arbitrarily across classes.

4. *State-dependent service*: The service time at a FCFS server can depend only on the total queue length at the server. The service time for a particular class at servers with PS, LCFS-PR, and IS discipline can additionally depend on the queue length of that particular class, but needs to be independent of the queue lengths of other classes. The aggregate service rate of a subnetwork can be dependent on the total number of customers in the subnetwork.

5. *Arrival processes*: Bulk arrivals are not permitted. Arrival rates may be state dependent. In open networks, the interarrival times of a class should be exponentially distributed. Networks can be mixed, i.e., open with respect to some classes and closed with respect to others.

Some real world behavior of computer systems does not satisfy the product form constraints, e.g, simultaneous resource possession, resource blocking, and the creation and synchronization of sub-processes [130]. It can be shown that individual queues in a product form or *separable* network can be analyzed independently of other queues [56]. An important property of product form queueing networks is a simple closed form expression of the stationary state distribution. In other words the joint probability of queue lengths of M queues can be computed by multiplying the individual probabilities. In general, the stationary joint queue length probability (sometimes denoted equilibrium probability) π defined by the solution of the associated Markov process as given in (2.8) has a product form solution:

$$\pi(n) = \frac{1}{G} V(n) \prod_{i=1}^{M} g_i(n_i), \tag{2.9}$$

where G is a normalizing constant and n is the total network population. The function V is defined based on the network parameters, i.e. arrival and service rates at the servers and routing probabilities, and g_i is a function of the state n_i depending on the type of server $i, 1 \leq i \leq M$. For open networks holds $G = 1$, while an attribute of closed networks is $V(n) = 1$ [46].

The basic input parameters of a queueing network are the number of servers, customer routing probabilities, scheduling disciplines, and service times. In addition, open models are parameterized with an arrival rate, while closed models take as input the population. Important output parameters are customer response times, as well as queue lengths, throughputs, and utilizations at the servers. Let T be the length of time the system is observed, I the number of customer arrivals, C the number of customer completions, and B the length of time the server was observed to be busy. All following basic quantities of queueing networks are assumed to be averaged [130]:

Arrival rate:	Rate at which customers arrive to the server	$\lambda = \frac{I}{T}$,
Throughput:	Number of completions per unit of time	$X = \frac{C}{T}$,
Utilization:	Server busy periods per unit of time	$U = \frac{B}{T}$,
Demand:	Service requirement per customer	$D = \frac{B}{C}$,
Response time:	Time customer spends at the resource queueing or in service	R,
Think time:	Time customer spends at a delay station	Z,
Queue length:	Number of customers at server including those in service	Q,
Arrival queue length:	Length of queue seen by customer at arrival instant	A.

In models where customers do multiple trips to the servers before they complete, it may be of interest to study the performance indices at the individual resources. The visit count V_i describes the average number of trips a customer makes to a server and is defined as:

$$V_i = \frac{C_i}{C}, \tag{2.10}$$

where C_i are the number of customer completions at the ith resource. It is important to note that the service requirement of a customer may be described in two ways. A customer may perform V_i visits to a server and at each visit requires the service S_i. Equivalently, the total demand of service at the ith server required by the customer may be specified as:

$$D_i = V_i S_i. \tag{2.11}$$

In addition, the total service demanded by a customer at all resources, denoted D, is defined as the sum of the D_i.

The analytical methods for performance evaluation of queueing networks can be categorized into operational analysis [62, 81] and stochastic approaches [56]. In addition to simulation, this thesis focuses on the analysis of queueing network models using some fundamental relationships derived by operational analysis [130]:

The Utilization Law: $U_i = X_i S_i = X D_i,$

Little's Law: $N = XR,$

The Response Time Law: $R = \frac{N}{X} - Z,$

The Forced Flow Law: $X_i = V_i X,$

where N is the number of customers in the system and U_i and X_i respectively denote the utilization and throughput at the ith server. The performance indices of product form closed queueing networks can be computed with a recursive algorithm called *mean value analysis (MVA)* [168]. This technique allows computation of the steady state queue lengths without evaluation of the normalizing constant G, described in (2.9). Consider a closed network with N customers and M load independent servers. The MVA algorithm is based on the following recursive relationship:

$$R_i(N) = S_i(1 + Q_i(N-1)), \tag{2.12}$$

where R_i and S_i respectively describe response time and service time at the ith server. The term $Q_i(N-1)$ stands for the mean queue length at server i with $N-1$ customers in the network. For an intuitive explanation of this recursion consider a tagged Nth customer added to a network with $N-1$ customers. Upon arrival at the ith server, the tagged customer sees $Q_i(N-1)$ customers ahead, including the one being served. The tagged customer therefore expects to wait $Q_i(N-1)S_i$ before being admitted to the server. Consequently the total response time of the tagged customer accounts to $(1 + Q_i(N-1))S_i$ which includes the service time of the tagged customer itself [113]. Given the response times at individual servers computed from (2.12) the steady state queue lengths can be readily derived applying Little's Law to the network:

$$X(N) = \frac{N}{\sum_{i=1}^{M} V_i R_i(N)}, \tag{2.13}$$

$$Q_i(N) = X(N) V_i R_i(N). \tag{2.14}$$

While the MVA algorithm provides exact solutions for product form closed QN, it is difficult to find approximations for *Fork-Join Queueing Networks (FJQN)* [56, 193]. This subclass of QN does not have product form properties and has been used to model multiprocessor systems, where parallel jobs can be split, i.e. *forked*, into tasks that are then concurrently executed on separate processing units. A job only completes once all tasks have been processed and

Table 2.1: Summary of Main Extensions to the JINQS Tool.

Class	*Description*
Package: `network`	
`AdmissionControlNode`	Controller for custom admission policies
`BatchCustomer`	Batch job
`BatchTunnelForkStation`	Job fork element
`BatchTunnelJoinStation`	Job join element
`LoadDependentQueueingNode`	Load dependent queueing element
`SFQCustomer`	Job with proportional priorities
`SFQQueue`	Queue with start-time fair queueing discipline
`SFQMergingQueue`	SFQ Queue with job merging functionality
Package: `tools`	
`LoadDependentDistributionSampler`	Load dependent service times

hence needs to wait for all tasks to finish and *join*. The *fork-join* structure is a simple model of the behavior of such parallel systems [194, 195]. However, the synchronization constraints introduced by forking and joining and the resulting complexity make it difficult to analyze these systems [193].

In the performance evaluation of complex systems it might be useful to decompose models into several submodels and solve them independently [121]. For example, if the model contains a number of classes of weakly interacting random processes it may be possible to consider each class separately. *Decomposition approximations* have been shown to be especially applicable to queueing networks [172].

For our simulations we build on the event-driven simulation tool JINQS [87]. The open-source Java implementation constitutes an extensible library for simulation of multiclass queueing networks. JINQS consists of the two Java packages `network` and `tools` which provide primitives for compositional assembly of queueing networks and modeling of advanced constructs, e.g., customer- and state-dependent routing, time delays, and simultaneous resource possession. Application specific behavior can be introduced via subclassing. Table 2.1 summarizes our most important extensions to the JINQS implementation.

2.3 Virtualized Systems

Software systems have traditionally been installed on dedicated infrastructures which offer full control over the underlying hardware. The call for more efficient utilization of hardware

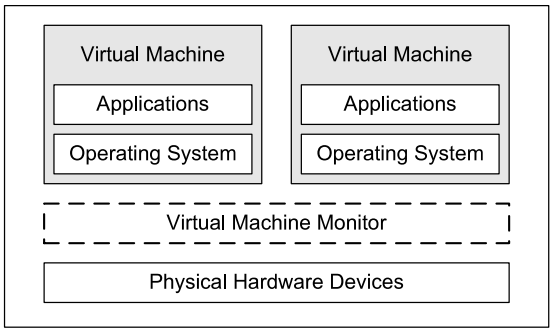


Figure 2.4: Example System Architecture Using Virtualization Technologies

resources, as well as for more agile and dynamic IT infrastructures has led to new developments in *Virtualization* technologies, which were initially developed as a method of time-sharing extremely expensive mainframe hardware [77, 94].

Figure 2.4 shows a conceptual illustration of a virtualized system architecture. In virtualization several isolated guest systems can share the underlying hardware by means of a software layer called *Hypervisor* or *Virtual Machine Monitor (VMM)*. The guest domains are independent entities running separate operating system instances, while the VMM manages the concurrent execution of the guest *Virtual Machines (VMs)*. The currently available system virtualization technologies can be categorized into *full virtualization*, *paravirtualization*, and *hardware assisted virtualization* [133, 173].

Full virtualization. These technologies allow hosting of VMs without modifications of the guest operating system kernel. The VMM virtualizes the hardware resources by exporting a functionally identical virtualized hardware interface that reflects the underlying machine architecture. Hardware instructions submitted by the guest system are trapped and managed at the VMM layer. This approach simplifies migration and portability, since the virtualization is fully transparent to the guest OS. However, the management of concurrently executing VMs can induce VMM overheads that may lead to variations in system performance [124, 203]. Full virtualization technologies are commonly used in enterprises and, for example, are championed by companies such as VMware [18, 20].

Paravirtualization. Management operations of the VMM layer may be reduced with specialized guest systems that directly utilize properties of the virtualized environment. In paravirtualization the VMs run modified operating systems that facilitate more efficient communication with the VMM [48]. The approach requires changes to the guest operating system kernel, which is a requirement difficult to implement for some proprietary software. An example platform using paravirtualization is *Xen* [19]. Related work has used this open source project for performance evaluation of virtualization overheads [73, 207].

Hardware assisted virtualization. Another approach to reduce VMM overheads and improve

performance is to incorporate support for virtualization directly in hardware architectures. In particular, the hardware exports a number of primitives to the VMM, e.g., for combination of control states for guest and host or direct execution of guest instructions without trapping [32]. For example, the open source project *Kernel-based Virtual Machine (KVM)* [5] is a platform exploiting hardware with virtualization extensions.

In this thesis we focus on the performance evaluation of systems using full virtualization.

2.4 Storage Systems

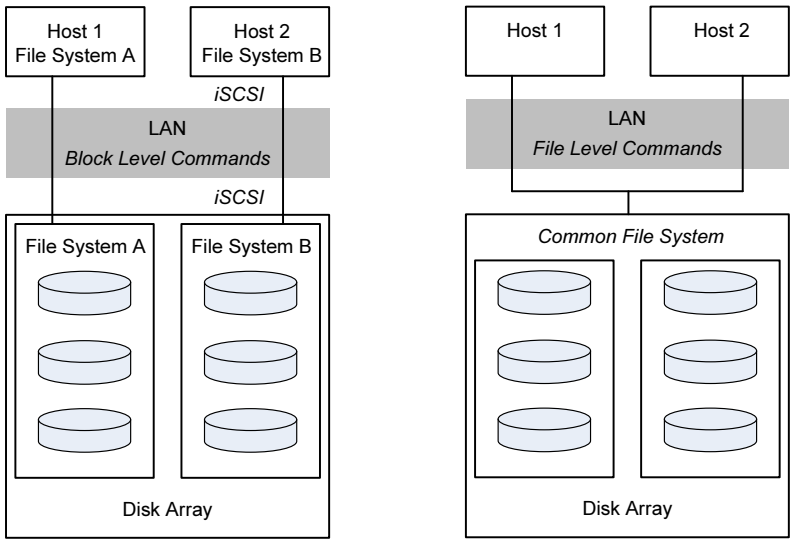
The term *storage system* has initially been used to describe *hard disk drive (HDD)* technology introduced by IBM in the early '50s. Since then the evolution of modern storage systems subsequently added layers of software and hardware to the raw capabilities of storage devices in order to build reliable and manageable systems with high performance [152]. This development has led to complex storage hierarchies in enterprises where storage tiers may comprise random access memory (RAM), solid state disks (SSDs), HDDs, and tape storage [33, 83, 209].

The communication between host side storage controllers and storage devices can be organized using file-oriented or block-oriented protocols [134]. In file-oriented systems the protocol provides a complete file abstraction, e.g., file creation, file deletion, file reads and writes. Conversely, block-oriented systems provide a physical device interface, e.g, for read and write requests to disk blocks. Thus in file-oriented systems the file system related functions are performed by the storage system, while with block storage these functions need to be managed by the host submitting data access operations [121].

For block storage the Small Computer System Interface (SCSI) protocol can be used to manage the communication between operating system hosts and disk devices. According to the SCSI client/server model hosts typically act as clients and disks act as servers, respectively denoted *initiators* and *targets*. Storage is presented to the host as a number of contiguous storage areas called *logical units (LUs)* identified by a *logical unit number (LUN)* [174].

Storage devices can be dedicated and directly attached to a single host or shared by multiple hosts over a network. Resource pooling using networked storage has several advantages over directly attached storage, e.g., higher flexibility, scalability, and data mobility [134]. Network storage technologies differ in terms of the network composition, e.g., *Fibre Channel* or *Gigabit Ethernet*, and the employed communication protocols [74]. The two main system architectures for sharing of network storage among multiple hosts are *storage area networks (SANs)* and *network attached storage (NAS)*, respectively applying block- and file-oriented protocols [74].

Figures 2.5 (a) and (b) show simplified illustrations of disk arrays connected via SAN and NAS topologies, respectively. Disk arrays such as *redundant arrays of inexpensive disks*



(a) SAN (b) NAS

Figure 2.5: Examples Storage Attached Network (SAN) and Network Attached Storage (NAS).

(RAID) [72] combine multiple independent disks into a single large unit: Data can be striped across multiple disks for improved performance or stored redundantly for better reliability. In Figure 2.5 (a) the host connects to the SAN over a *local area network (LAN)*. The communication is organized using the *internet SCSI (iSCSI)* protocol which essentially encapsulates SCSI commands for transport over TCP/IP networks. In the NAS example, depicted in Figure 2.5 (b), all disks in the array share a common file system and are accessed using file system commands.

In the first part of this thesis we evaluate the performance of data access operations to a shared block storage device. In particular, we consider a fully virtualized server connected to a SAN. The VMM hosts a number of VMs that share disk space located on the same disk array. Figure 2.6 shows a high-level illustration of such a virtualized system. The *virtual* disks of the VMs are represented by files that reside on the same LU. The VMM traps data access operations of the VMs and manages disk I/O using the iSCSI protocol [24]. The disk array is governed by a storage server through a hardware RAID controller.

2.5 In-Memory Databases

Database systems are an essential part of the enterprise IT stack as they are used to maintain business critical data, e.g, of clients, products, accounts, and employees. The workloads of enterprise databases typically consist of *online transaction processing (OLTP)* or *online analytical processing (OLAP)* queries. OLTP queries comprise short running transactions that

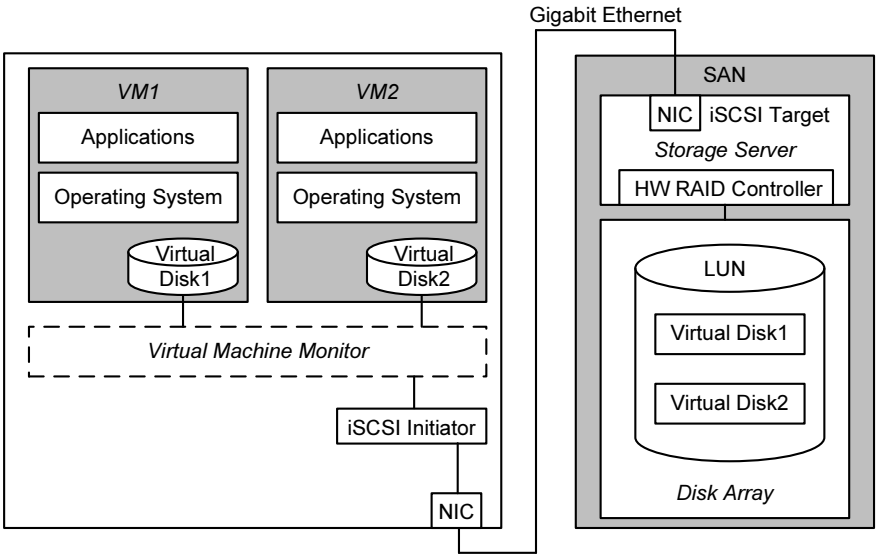


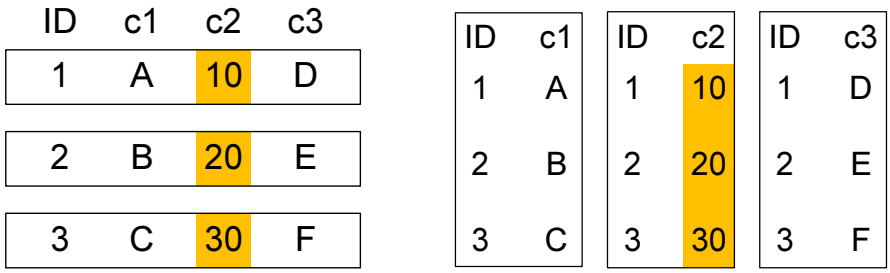
Figure 2.6: Example Virtualized System Connected to a SAN

access only small portions of the data set and process both read and write operations, e.g., sales order entry or banking transactions. Conversely, OLAP workloads consist of long running queries performing mostly read operations on large data sets. For example, *Business Intelligence (BI)* applications submit OLAP queries for generation of sales reports. A common storage medium for hosting of enterprise database systems are disk arrays where query performance is dominated by the time consuming I/O operations to secondary storage [136]. Main memory database systems can significantly decrease the data access times of OLTP and OLAP workloads by maintaining data sets in system RAM [122, 164].

In-memory databases were initially proposed in the ’80s [82] and have recently evolved as a mainstream technology. Some of the first commercial in-memory systems for OLTP workloads were TimesTen [7] (acquired by Oracle), P*TIME [70] (acquired by SAP), and SolidDB [4] (acquired by IBM). Examples of in-memory systems for OLAP workloads are the research project Monet [137] and the SAP development TREX [131]. In addition, related work proposes in-memory databases for mixed OLTP/OLAP workloads [122, 164].

Relational databases store data in structures where rows correspond to real-world entities and columns correspond to entity attributes. For physical data storage this conceptual two dimensional data organization needs to be mapped into one dimensional structures as storage media provide only one dimensional interfaces, i.e., read or write from a given linear offset. Databases implement this mapping by storing table structures either row-by-row or column-by-column as depicted in Figures 2.7 (a) and (b), respectively. The choice of implementation depends on the application scenario and performance expectations [28].

Column stores [76] maintain separate structures for each attribute and preserve the logi-



(a) Row Store (b) Column Store

Figure 2.7: Example Row and Column Store Databases.

cal table schema using surrogate identifiers. The performance of OLAP workloads can be significantly increased using this approach [55, 175], since analytical applications are mostly attribute-focused as opposed to entity-focused [28]. In other words, only a small number of table columns need to be considered for satisfaction of OLAP queries.

Column store databases can increase the performance of OLAP workloads via more efficient data access and processing operations. The attribute-focused nature of the workload allows reading of only the required columns resulting in more cache-friendly I/O patterns. Thus, in the case of disk storage only few seeks are required to find the page of a block that has the first field in a particular column, and then data can be read in larger blocks. For in-memory storage the pre-fetching algorithms of DRAM controllers can exploit spatial locality and ensure good L2 cache hit ratios for sequentially laid out columns [175]. In addition, compression techniques can be used to reduce the volumes of processed and cached data allowing more efficient numerical processing and caching strategies [131]. For example, the data sequence “eeeee” might be compressed to “e{5}”.

The processing of column store data facilitates exploitation of high parallelization levels. Table attributes may be partitioned into individual fragments and distributed to multiple processors. The illustrative example depicted in Figure 2.8 shows multiple threads processing individual table columns and regions. OLAP workloads can especially benefit from the parallelization capabilities of modern multicore architectures, since the parallel execution of read-only queries requires less synchronization effort [175].

In this thesis we study the thread parallelization levels of a system hosting a column store database in order to derive novel admission control policies for database queries. Our techniques are especially applicable for in-memory databases as they ignore the I/O subsystem and focus on CPU processing tasks.

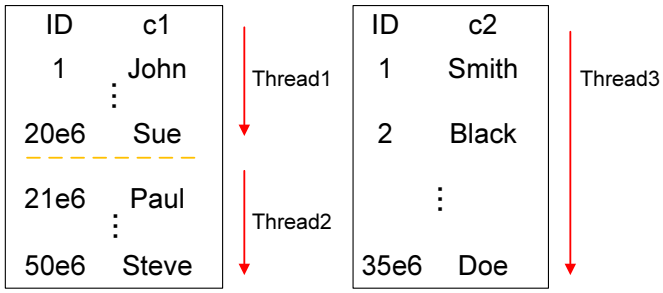


Figure 2.8: Example Column Store Parallelization

2.6 Summary

This chapter has introduced the preliminary concepts used throughout the course of this thesis. In the first part of the thesis we use benchmarks for emulation of disk workloads to evaluate the performance of disk I/O requests when submitted to a shared block storage device. In particular we consider multiple VMs hosted on the same fully virtualized host connecting to the storage device via block-oriented protocols. The second thesis part studies the performance of a decision support type workload on an in-memory database system. In particular we propose admission control policies for database queries that consider the resource usage patterns of a query mix concurrently running on a server with multiple processing cores. For performance evaluation we make use of the concepts provided by queueing theory. The presented open and closed models are solved with operational analysis and customized simulation tools. Model parameterization is derived from measured system traces.

Part I

Performance Models of Storage Contention in Cloud Environments

3 I/O Performance Prediction in Homogeneous Consolidated Environments

3.1 Introduction

The performance of I/O-bound applications is dominated by the time required by the operating system to schedule read and write operations and by the response times of the storage devices in completing such requests. Since changes in the workload, as well as in the software and hardware environments, can affect the latency of disk I/O requests, it is often useful to define performance models to anticipate the effects of a change. This is especially important in virtualized data centers, where the concurrent shared use of a storage device by several Virtual Machines (VMs) managed by a Virtual Machine Monitor (VMM) can lead to significant performance degradation [124]. In such systems estimates of I/O contention for a given VM placement configuration can support management and consolidation decisions. However, modeling the performance of disk requests is very challenging due to the joint interaction of the I/O flows issued by several VMs and because of the complexity of caching mechanisms, scheduling algorithms, device drivers, and communication protocols employed by both the VMs and the VMM.

We tackle this complexity by introducing a trace-driven simulation approach for I/O performance prediction in consolidated environments where multiple VMs can share access to a remote storage server. Our methodology, summarized in Figure 3.1, requires first a study of VMs when they run in isolation on a virtualized server. After collecting measurements, we use simulation and specialized parameterization techniques to forecast the impact of consolidation on I/O performance.

Specifically, for each VM of interest we collect traces of arrival times and estimated service times for I/O requests. We then use these isolation scenario traces to parameterize a queueing model for a specified consolidation scenario. A feature of our queueing model is also the use of start-time fair queueing (SFQ), a popular implementation of fair share scheduling which is adopted in VMMs. The model focuses on the prediction of response times for homogeneous workload consolidations, i.e. scenarios where multiple VMs submit workloads of the same type, and hence is denoted Homogeneous Model.

Motivated by the fact that a trace-driven simulation that uses only such arrival and service time traces typically *fails* to predict accurately I/O request response times under consolidation, we define an iterative algorithm for optimal parameterization of the simulation model. Specifically, the algorithm estimates the performance impact of VMM level I/O optimizations such as the splitting of I/O requests and offers a search technique for model calibration. Ex-

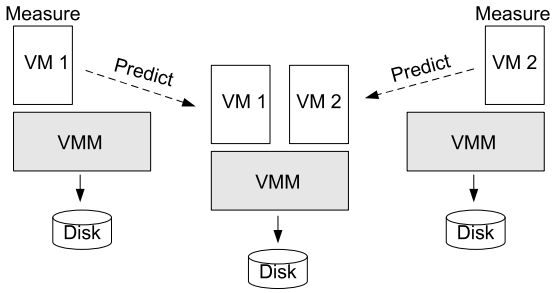


Figure 3.1: Problem Approach.

tensive experimentation on test workloads generated with the Postmark (PM) and FFSB disk benchmarks reveals that our methodology can forecast successfully consolidation effects on I/O performance.

Summarizing, the main contributions of this chapter are twofold.

1. Our methodology allows us to parameterize a simulation model based on data obtained inside VMs in isolation experiments. It requires very little information from the VMM thereby effectively treating the VMM as a black box. It also obviates the need to collect model training data for different VM consolidation scenarios.

2. The trace-driven simulation model is enhanced with an iterative calibration technique that approximates parameterization inaccuracies in the absence of detailed VMM level measurements.

The remainder of this chapter is organized as follows. Section 3.2 lists the addressed research questions. Section 3.3 motivates the use of prediction tools to quantify performance degradation of disk requests in shared environments. Related work is shown in Section 3.4 and Section 3.5 introduces the reference system and monitoring tools for our study. The proposed Homogeneous modeling methodology is presented in Section 3.6 and its validation results are shown in Section 3.7. Section 3.8 offers summary and conclusions.

3.2 Research Questions

In this chapter we address the following research questions:

- Is it possible to accurately predict the performance of a fully virtualized system with a queueing network simulation model?

- Can we use block device I/O traces obtained within VMs in isolation benchmark experiments for parameterization of models simulating homogeneous consolidation scenarios?

- How can we calibrate the simulation model given the black box view of the VMM?
- How does the quality of such a trace-driven simulation model relate to an existing technique for performance prediction?

3.3 Motivational Example

In virtualized environments, the VMM adds an additional layer to the I/O architecture that needs to be considered in I/O performance models. The hypervisor supports storage requests from varying operating system types and the disk scheduler might do some further optimization processing of incoming traffic. For example, disk schedulers do not operate in a true first-come-first-served (FCFS) manner, but try to optimize disk access patterns by reordering queued requests according to the targeted disk sector addresses. Furthermore, large requests can be split into smaller requests while similar requests that arrive within a specific time window can be aggregated and processed together in order to decrease latencies of the storage device.

In consolidation scenarios where more than a single VM submits large numbers of I/O requests, competition for the disk drive can lead to significant increases in latencies, i.e. response times. To illustrate the problem Figure 3.2 compares the mean response times in three experiments conducted on our reference system, which is introduced in Section 3.5. In the first two experiments, denoted isolation experiments, single VMs run on the system thus avoiding contention from other VMs. In the third experiment, denoted consolidation experiment, we run two VMs on the same virtualized server and find that their disk requests experience storage contention. Response times are computed as the time between request issue and request completion as recorded by the VM operating system.

In consolidation we record large mean response time increases of roughly 170% and 71% for web and mail server type disk workloads, respectively. This makes a strong case for the significant performance impact that consolidation can have on end-to-end response time of I/O requests and motivates the investigation in this thesis. Specifically, the example provides motivation for the need of accurate models that can capture and quantify the possible I/O performance degradation related to consolidation effects.

3.4 Related Work

This section outlines the related work in the area and is structured into the categories scheduling, measurement, and prediction.

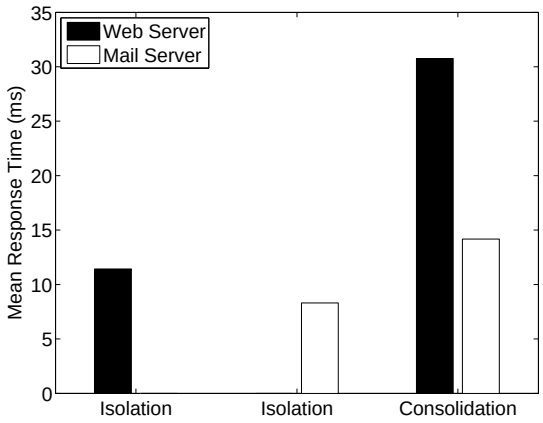


Figure 3.2: Effect of VM Consolidation on Disk Request Latencies.

3.4.1 Scheduling

A large amount of research literature is concerned with scheduling algorithms for disk I/O in virtualized environments. The main challenges regarding scheduling are to provide fair access to the shared storage resource for all applications, while maintaining performance isolation, i.e. disk accesses by one application should not affect the I/O performance of another. This work can be structured based on the system layer under investigation. The authors in [58] use Xen and VMWare Server environments to investigate how the combination of VM and VMM level disk schedulers affects throughput and fairness between VMs. In [123] the performance isolation between multiple in-VM applications hosted on Xen is studied and extensions to current Linux schedulers are proposed. Other approaches investigate the performance impact of scheduling algorithms at the VMM layer. In [158] multiple VMs are consolidated on a Xen server. Each VM submits a single processor-intensive, bandwith-intensive, or latency-intensive workload and the authors propose enhancements to the VMM I/O scheduler to assign resources more fairly. A virtual I/O scheduler providing performance isolation and fair I/O resource sharing with concurrent disk access is developed in [180]. Furthermore, the PARDA [100] technique allows coordination of I/O scheduling across multiple independent virtualized servers sharing a storage device [100]. In contrast, our work focuses on the prediction of I/O request performance as observed by the operating system within the VM and considers only a single VMM instance. Furthermore, we do not propose changes to the system's I/O schedulers or evaluate the fairness provided by the reference system. Instead we use the standard system configuration and aim to predict the performance of consolidated storage workloads. Throughout our work we assume a system configuration assigning equal proportional storage resource shares to all VMs.

Other research addresses the *Quality-of-Service (QoS)* during shared resource usage. The work includes approaches for proportionate bandwidth allocation to storage servers for appli-

cations deployed on a single server [114] or on multiple independent servers [99, 104]. In addition, techniques guaranteeing disk request latencies have been proposed [103]. The QoS is out of scope of our work.

3.4.2 Measurement

The performance of a VM may vary over time depending on the background workload, i.e. interference, originating from other VMs deployed on the same host. Several research groups have provided measurement studies of interference in virtualized Xen environments. In [124] the interference effects between system-level workloads, e.g. *cp* and *make*, are quantified using a normalized score. The authors consider a two VM consolidation scenario where each VM submits a single workload and find a significant degree of performance interference between disk I/O-intensive applications. A monitoring framework for measurement of CPU overheads in the VMM layer is presented in [73]. The workload consists of a single VM and disk and network I/O intensive applications. Similarly, performance interference due to the concurrent processing of network I/O workloads is measured in [165] and [143] for two and up to seven VM consolidation scenarios, respectively. The performance of multi-tier applications in varying deployment configurations and consolidation scenarios is measured in [161]. Closer to us, the workload characterization studies in [34] and [101] measure disk workload characteristics and performance metrics in isolation and two VM consolidation VMware environments, respectively. Contrary to us they do not consolidate by placing two workloads on the same LUN, but consolidate two LUNs into a single RAID group.

3.4.3 Prediction

Efficient resource sharing requires some capacity planning and workload analysis in order to identify which workloads to consolidate on the same server. Predictive performance models have been shown to be useful for assisting workload placement. In [207] a regression model is proposed to predict the CPU resource requirements when migrating applications from a native to virtualized Xen environments. The authors in [51] discuss asymptotic bounds and average value equations for open class queueing networks for a similar use case. In [35] a mathematical model for prediction of disk I/O throughputs is derived when moving from a native system to an isolated VMware ESX server environment. In contrast, our predictive models focus on varying numbers of consolidated VMs and do not evaluate overheads compared to native installations.

The dynamic allocation of shared resources requires a provisioning technique determining how to partition server resources between the applications. In [71] a number of applications are configured to share a server. The server is modeled as a time-domain queueing model and

a non-linear optimization technique is used to compute and assign resource shares considering workload variations over time. Other approaches use *Layered Queueing Networks (LQNs)* to model the consolidated deployment of multi-tier applications in virtualized environments and utility functions for resource allocation [117, 118]. The authors in [52, 144] employ a combination of open and closed queueing models and utility functions to dynamically allocate CPU resources. Instead, our work addresses systems in steady state and does not consider dynamic allocation.

The authors in [124] develop mathematical models using *principal component analysis* and linear regression techniques for prediction of normalized performance scores in two VM consolidations. In [145] a statistical model for estimation of joint CPU demands of consolidated VMs is proposed. Recently, the authors in [127] present an iterative model training technique based on artificial neural networks for resource allocation in consolidated virtualized environments. While some of the work above captures coarse grained disk requirements in the model in order to predict effects of resource allocation changes on performance of consolidated applications, none specifically tries to predict fine grained disk I/O request performance degradation due to workload consolidation. Closer to us, prediction of disk request response time granularity based on a machine learning technique is presented in [200, 201]. The approach employs Classification And Regression Tree (CART) models and treats the storage device as a black box. However, the model does not consider shared resource access or workload consolidation.

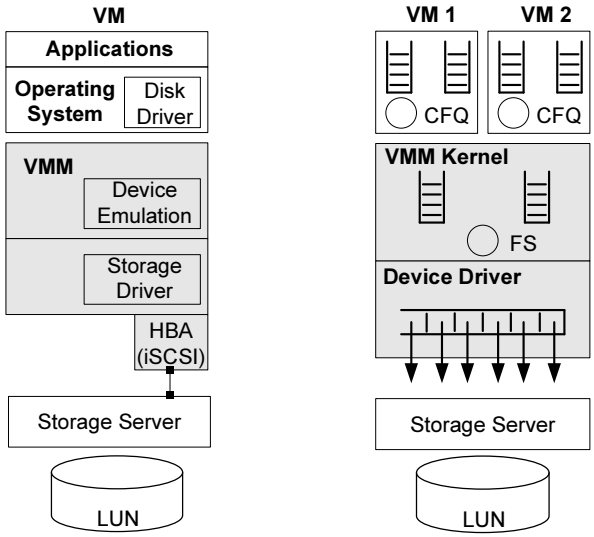
3.5 System Characteristics

We begin with a specification of the hardware and software environment used in the experimentation and advance to present the tool used to obtain I/O measurements. Our modeling techniques have been tested only on the architecture described below, but we believe them to be representative of virtualized environments which adopt similar software technologies.

3.5.1 Reference System

We conduct our study on an AMD-based enterprise server with 4 quad-core processors containing a total of 16 CPU cores each clocked at 2.21GHz. The system is equipped with 68GB of RAM and is connected to an OpenFiler [6] storage server via the iSCSI protocol and 1 GBit Ethernet. The storage server comprises 8 Intel Xeon processors clocked at 2GHz and 15GB of RAM. It manages a 3ware 9000 series SATA-II hardware RAID controller with 256MB of RAM. The 15 disc RAID 5 array uses a stripe size of 64k and disk write-caching is turned off.

On the server we host the virtualization platform VMware ESX Server 3i - 3.5.0 [18], which accesses the storage device through a software iSCSI host bus adapter (HBA). We specifically



(a) Storage Subsystem

(b) Storage Queues

Figure 3.3: I/O Architecture and Storage Queues of the Reference System.

choose VMware, as it is a widely adopted virtualization technology. The virtualized environment consists of multiple Debian 5.0.1, kernel 2.6.26-2, guest VM systems, each with identical configuration of 1 virtual CPU, 500MB RAM, and 50GB of "virtual" hard disk formatted as *ext3* file system.

The virtual disks are represented by a large file and can be thought of as a linear array made up of units of space, i.e. logical blocks. In the remainder of this paper the term "block" refers to logical block units, rather than the storage device's physical block units.

The VMM formats and stores virtual disk files in the Virtual Machine File System (VMFS). The VMFS provides distributed locking mechanisms facilitating concurrent access to virtual machine disk files from multiple physical machines. A VMFS volume may also be extended over multiple LUNs for storage pooling [24].

The storage contention in our reference system is characterized by a single connection from the ESX software iSCSI HBA to a single LUN on the RAID array. We do not consider scenarios with multiple physical hosts or VMFS volumes spanning over multiple LUNs. All virtual machine disk files are hosted on the same LUN.

Disk I/O requests issued from a VM consist of one or multiple contiguous blocks for either reads or writes. Once an application running inside the VM submits a request, the request first goes to the disk driver of the VM operating system as shown in Figure 3.3 (a). The driver processes the request and forwards it to the VMM, where it is trapped and may undergo further optimization operations before being issued to the LUN via the storage driver and the iSCSI

HBA [35].

On their way through the previously described layers of the storage subsystem, requests can be queued multiple times, as illustrated in Figure 3.3 (b). Hence latencies of requests may be affected by multiple queueing delays. Furthermore, requests may undergo multiple optimizations such as aggregating and reordering operations by in-VM schedulers, as well as request splitting at the VMM level. Such operations are used to optimize disk access patterns, e.g., by aggregating multiple requests for small amounts of contiguous blocks to fewer requests for large amounts of contiguous blocks. In virtualized environments these operations can have a significant impact on disk request performance, as scheduling policies at VM and VMM level may impair each other [58].

In our environment the Debian guest VMs are configured to use the completely fair queueing (CFQ) [45] scheduler, which has per default 64 internal queues to maintain and keep disk I/O requests [21]. The in-VM scheduler optimizes disk access for the ext3 file system, which is configured at the default block size of 4kB. Hidden from the operating system of the VM, the VMM conceptually comprises two sets of queues, namely the VMM kernel queues and the device driver queue. The VMM kernel maintains a queue of pending requests per VM for each target SCSI device [100], controlled with a fair-share (FS) [79] scheduler. This class of algorithms schedules shared resources among competing flows of request classes where each request is associated with a cost [53, 95, 98]. Resource sharing entails the need to control how consolidated request classes consume the resource in order to isolate performance and manage differentiated resource access. Without this management load surges of competing flows may impair each other unacceptably. FS allocates resource capacities in proportion to weights that have been assigned to the competing request classes. See Section 3.6.1 for more detailed information on the scheduling.

Furthermore, the server maintains a device driver queue for each LUN, which controls the *issue queue length* defined as the number of pending requests the server can have at the storage device at a time [11]. Virtualized servers typically allow configuration of the issue queue length for each LUN [58, 99]. When multiple host servers issue requests to the same LUN, this parameter can be used to control resource utilization and fairness across hosts.

Summarizing, our reference system comprises a network of interconnected queues with multiple classes of customers corresponding to the multiple VMs and various scheduling disciplines, which need to be represented in the performance model. In this case the application of classic solutions for product form queueing networks [49] is complicated due to complex splitting, aggregating, and reordering operations on arriving requests that depend on spatial locality on the storage device. Although forking and joining operations may alleviate such difficulties, they are hard to parameterize in the absence of direct measurement of the VMM internal optimization processes, which is the typical situation in practice. Furthermore, the batched sub-

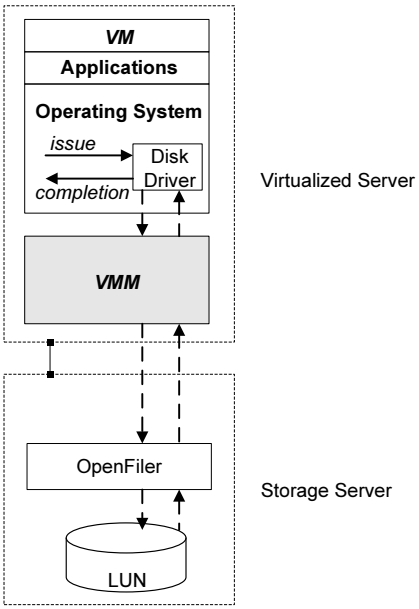


Figure 3.4: Monitoring of Reference System With the Blktrace Tool.

mission of requests can result in extreme behavior of the arrival patterns where large amounts of requests are condensed into large bursts. Bursts have been identified as important sources of performance degradation [67,147] and we handle them in this work by resorting to trace-driven simulation.

3.5.2 Measurement Tool

This section describes the monitoring tool we have used to quantify the performance of disk requests, as well as to collect necessary data for parameterization of our model. All measurements are obtained inside VMs with the block layer I/O tracing mechanism *blktrace* [1]. Figure 3.4 illustrates how the tool allows us to capture disk I/O traces, as they are recorded from the VM operating system kernel. Traces comprise *issue* and *completion* events for each disk request: *Issue* events specify that a request that has previously resided in the disk I/O scheduler queue of the VM operating system has been sent to the driver, while *completion* events mark that a previously issued request has been completed [61]. Collecting this data at the operating system level, rather than at the level of the in-VM application, has the advantage that sorting and merging operations of the VM CFQ scheduler are reflected in the monitoring trace.

We prevent monitoring data from interfering with measurements by storing captured events into a temporary file system residing in system RAM [25]. Figure 3.5 shows an example trace comprising a relative nanosecond scale time stamp for each event, the event type, a flag that indicates whether a request was a read or write, the logical block address (LBA) pertaining to

TIME	EVENT TYPE	READ/WRITE TYPE	LBA	BLOCKSIZE
...				
11.575226201	I	W	79930319	8
11.576754924	C	W	79909447	8
11.576766766	D	W	79910559	32
11.576838516	D	W	79910695	32
11.576924596	D	R	79910743	32
11.577008319	D	R	79910871	16
11.577093686	C	W	79909463	8
...				

Figure 3.5: Example Disk I/O Trace From Blktrace

the request, and the number of blocks accessed. The blktrace tool can collect a large number of events, e.g., inserted into the I/O scheduler queue (I), issued to the driver (D), and completion of the previously issued request (C). This detailed information is helpful for building the system model, since it enables us to compute per request arrival times at the VMM. Based on the unique LBA identifier we are able to correlate events corresponding to the same request. We note that we calculate per request response times as the time difference between completion and issue events in the trace. We are aware of the timekeeping inaccuracies in virtualized environments and use the network time protocol (NTP) to synchronize time between VMs. NTP has been reported to work fairly well on our reference system [22].

3.6 Homogeneous Model

Our model comprises trace-driven simulations to predict the performance degradation of VM disk request response times due to storage device contention in homogeneous workload consolidations, i.e. scenarios where multiple workloads of the same type are deployed on a single server. The fundamental idea of our approach is to first record application traces in isolation benchmark experiments and then utilize these traces to parameterize simulation models of consolidation scenarios. The challenge is to define realistic models of contention and of the internal VMM I/O optimizations which affect the end-to-end delays of read and write operations. Our methodology considers a heuristic for request splitting behavior at VMM level and introduces a model calibration technique to alleviate parameterization inaccuracies, e.g. on request service times. Model parameterization is solely based on measurements obtained within the VMs and information that we gathered from available documentation on VMM operations such as fairshare scheduling and splitting. As a result, we essentially treat the VMM as a black box in this work.

Figure 3.6 shows an overview of our methodology and introduces some of the terminology

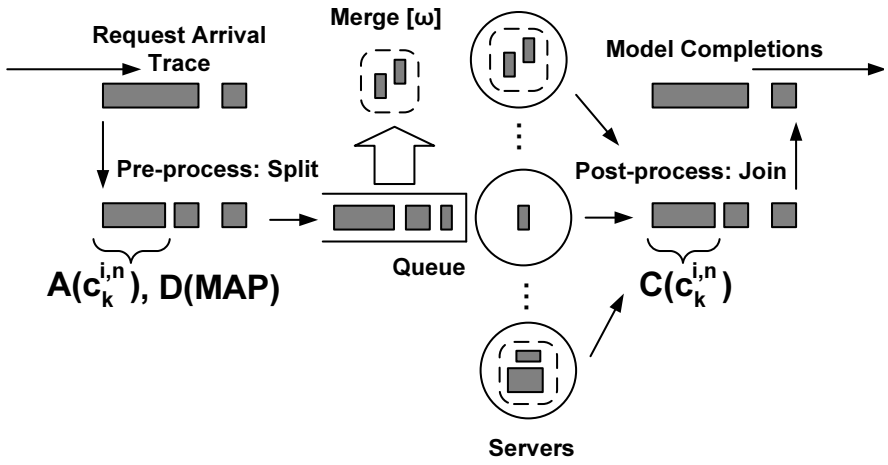


Figure 3.6: Methodology Overview Homogeneous Model.

used in the following sections. The model allows us to study a specified consolidation scenario consisting of a set of VMs concurrently sharing a storage device. Input parameters for the queueing model are request arrival traces obtained from in-VM measurements in isolation benchmark experiments for each of the VMs considered in the consolidation scenario. Our methodology can account for environment specific, VMM level request splitting behavior by means of a pre-processing step on arrival traces, where each i^{th} arriving request of class k, denoted c_k^i, may be split into n requests. Section 3.6.2.1 explains in detail how request arrival times $A(c_k^{i,n})$ are assigned. Requests are also provided with a service time $D(c_k^{i,n})$ sampled from a Markovian Arrival Process (MAP) as described in Section 3.6.2.2. As shown in Figure 3.6, we use a simulation model that schedules requests using the SFQ(H) [114] scheduling policy across a pool of servers, each representing a parallel connection to the LUN. This model additionally introduces a calibration technique by means of a merging heuristic. The merging bundles a configurable number of ω requests and enables them to share a server. The queueing network and the scheduling algorithm are presented in 3.6.1. The search algorithm we developed to iteratively estimate the parameter ω is introduced in Section 3.6.3.2.

Finally, the model outputs requests with a response time estimate $C(c_k^i)$. This involves a post-processing task wherein, as shown in Figure 3.6, the requests that have been split in the pre-processing step are rejoined.

3.6.1 Simulation Model

We represent the system under study as a multiclass open queueing model. Requests submitted from individual VMs are distinguished in separate classes as shown in Figure 3.7. As described in Section 3.5.1, virtualization environments typically provide a configurable parameter which

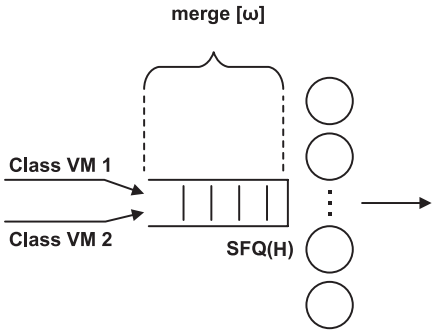


Figure 3.7: Queueing Model.

controls the maximum VMM issue queue length to the LUN. We capture this aspect by modeling the storage device as a pool of parallel servers. In our reference system this parameter is maintained at its default value of 32 and consequently our model comprises 32 servers.

The model implementation extends JINQS [87], a library for simulating multiclass queueing networks. Based on available documentation on the VMM, we implemented a practical SFQ disk scheduling discipline to schedule requests on to the servers. Fair queueing [79] algorithms are work-conserving and schedule shared resources between competing requests by allocating resource capacity based on proportional weights. Practical fair queueing algorithms constitute approximations to generalized processor sharing (GPS) [162] scheduling by considering requests to be indivisible, rather than fluid flows.

Conventional SFQ schedulers do not consider concurrent service of requests at a resource. Our simulation model implements SFQ(H), a special variation of SFQ [98], which has previously been used in related work [100] to model our reference system. The depth parameter *H* controls the number of concurrent requests in service and consequently corresponds to the number of servers in our model. Upon arrival of the i^{th} request c_k^i of class k, it is assigned a *start tag* $S(c_k^i)$ and a *finish tag* $F(c_k^i)$ by the scheduler. The tag values represent the times at which each request should start and complete service according to a system notion of *virtual time* $v(t)$. Tags are computed as:

$$S(c_k^i) = \max\{v(A(c_k^i)), F(c_k^{i-1})\}, \quad i \geq 1 \tag{3.1}$$

$$F(c_k^i) = S(c_k^i) + \frac{d_k^i}{\phi_k}, \quad i \geq 1 \tag{3.2}$$

where $A(c_k^i)$ is the arrival time of request c_k^i, $F(c_k^0) = 0$, $v(0) = 0$, d_k^i is the service time of the request, and $\phi_k > 0$ is the weight or share of class k, $\sum_{k=1}^{K} \phi_k = 1$. Throughout the

experiments, we assign equal shares to all request classes. The scheduling algorithm reserves the specified minimal share to each class. In cases where a class has no active requests surplus resources are shared among active classes according to their relative weights.

The scheduler issues a maximum of *H* requests to idle servers in increasing order of *start tags*. When a request completes the queued request with *min(start tag)* is selected and issued to an available server to maintain a concurrency level of *H*. *Virtual time* advances by assigning it the start tag of the last request issued on or before time *t*, i.e., the queued request with the lowest start tag at the time of the last issue. As mentioned previously, in addition to SFQ(H) scheduling we also select ω requests, merge them together, and issue the merged request to the servers. The superposition of this behavior with SFQ(H) is described in Section 3.6.3.2.

3.6.2 Model Parameterization

This section describes how we obtain the interarrival and service times of requests. In this step, we account for the request splitting operations of the VMM which are triggered when the block sizes of arriving requests exceed a certain threshold.

3.6.2.1 Interarrival Times

The simulation model is parameterized with measured arrival traces, which are recorded in a series of benchmark experiments. Benchmark workloads are submitted from within VMs running in isolation, where only a single VM is running on the server. For each VM we build a trace repository comprising multiple benchmark runs. Traces are recorded with the *blktrace* tool, where we include every in-VM request issue as an arrival in the model.

When predicting request response times for consolidation scenarios, we randomly choose an arrival trace for each considered VM from the repository and run a consolidation simulation. Parameterizing the simulation model with arrival traces measured in isolation experiments is a valid approximation, since our measurements in Section 3.7.2.2 show that the interarrival time distribution of disk requests to the VMM is not significantly impacted by workload consolidation. This indicates that in the presence of contention delays disk requests are queued at the VMM, rather than at the VMs. Queueing requests at the VMM is preferable, since queue depths should be larger than at the VMs and the VMM can use system specific information to optimize disk access of queued requests. We expect this observation to hold unless the VMM queues are saturated.

We account for splitting operations of the VMM by performing additional processing steps on model input parameters and output results. Each arriving request c_k^i has an arrival time $A(c_k^i)$ and a block size $B(c_k^i)$. Given the maximum request size threshold l_{max}, we pre-process

the trace and split all arrivals where $B(c_k^i) > l_{max}$ into N separate arrivals, such that:

$$N_k^i = \left\lceil \frac{B(c_k^i)}{l_{max}} \right\rceil \tag{3.3}$$

$$A(c_k^{i,n}) = A(c_k^i) \tag{3.4}$$

$$B(c_k^{i,n}) = \begin{cases} l_{max} & n \in \{1..N_k^i - 1\} \vee \\ & B(c_k^i) \text{mod } l_{max} = 0 \\ B(c_k^i) \text{mod } l_{max} & n = N_k^i \wedge \\ & B(c_k^i) \text{mod } l_{max} \neq 0, \end{cases} \tag{3.5}$$

where N_k^i is the total amount of splitting operations for arrival c_k^i determined by the ceiling function, mod finds the remainder of the division, and $n \in \{1..N_k^i\}$. Since splitting operations are performed by the VMM and are not visible to the VMs, previously split requests need to be rejoined once they have completed service. Our methodology includes a post-processing step on requests that leave the simulation model and as a result are assigned a response time $C(c_k^i)$. We compute the response times of joined requests as the mean:

$$C(c_k^i) = \frac{1}{N_k^i} \sum_{n=1}^{N_k^i} C(c_k^{i,n}). \tag{3.6}$$

As requests are likely to be served in parallel by the storage array, computing the mean response time of split requests can only be an approximation of the real behavior at the device. We investigate the effectiveness of this approximation in Section 3.7. In our reference system the VMM splits arriving requests exceeding a size of 512kB, corresponding to 128 blocks in an ext3 file system with 4kB block size. We consider this behavior in our model and set $l_{max} = 128$.

3.6.2.2 Service Times

Service times are key parameters for specifying queueing models and are typically estimated based on direct data measurement and statistical inference. A common approach to characterizing the resource consumption of requests is to monitor system utilization and use regression techniques based on operational laws [130]. As our black box approach does not involve instrumentation of the VMM or of the storage server in order to collect utilization samples, a practice which is in any case difficult or impossible in many real-world systems, we approximate the

service times of disk requests from response time measurements in isolation benchmark experiments. When utilization levels at the VMM are low, disk requests do not face queueing delays as they get instantly issued to the storage device. In fact our measurements show that the mean number of requests in the system during an isolation run does not exceed the number of available connections from the VMM to the LUN. Thus measured request response times in isolation should be a reasonable approximation to the actual service requirement at the disk.

Request service times belonging to a specific VM collected during isolation experiments are fitted to a MAP [69], which is then used to randomly sample a service time for each arrival. The role of MAPs in our methodology is to generate random traces which follow the same statistical properties (distribution, autocorrelations) observed in the isolation experiments.

3.6.3 Model Calibration Methodology

Our simulation experiments in Section 3.7.3.2 indicate that prediction results can be significantly improved if the model is calibrated with a merging heuristic to account for inaccuracies in parameter approximations. Calibration methods can aid in building simple models of complex systems [50], e.g., in case studies like ours where detailed documentation of commercial software internals are not available. As we mentioned previously, our model allows the scheduler to merge queued requests and use the merging to compensate for possible deficiencies in the parameterization of the model. For example, deficiencies in service times may arise due to the burstiness inherent in disk workloads and the resulting impact on storage system behavior [169]. By enabling the scheduler to merge, i.e. aggregate, queued request we effectively model such batching behavior. Furthermore, our analysis of storage server monitoring data in Section 4.5.2.1 shows evidence that, under simplifying assumptions, service times at the disk can be load dependent. For example, service times can decrease at high loads. Since we approximate service times from low load measurements, the merging of requests may help modeling of such load dependent behavior. The merging algorithm is described in Section 3.6.3.1. The iterative technique to quantify the number of scheduler merging operations performed for a given workload configuration is described in Section 3.6.3.2.

3.6.3.1 Request Merging Algorithm

The model merges a configurable number of queued requests in such a way that they still remain separate entities, but share a server, i.e., connection to the LUN. As shown in Algorithm 1, we pass a merge value parameter, denoted ω, to the simulator, which serves as a modeling abstraction in our black box model. Merging values can range in $[1, \infty]$, where we consider a merge value of 1 as a single job being issued per service station, i.e. no merging, whereas a large ω indicates that several requests are merged together before issue to the server. Since

Algorithm 1 Implementation of Merging

```
ω ← merge value
merged_jobs ← struct
while service station idle AND job queued do
  x ← get_int_value(ω)
  for i = 1 to x do
    job ← queued job with min start_tag
    if i == 1 then
      merged_jobs ← merged_jobs + job
    else
      if class job == class merged_jobs then
        merged_jobs ← merged_jobs + job
      else
        break
      end if
    end if
  end for
  schedule merged_jobs to idle service station
end while
```

requests are indivisible, we implement a function get_int_value to obtain from ω the number of merged requests in each round of calibration. For example, if ω is configured as 2.5 there is an equal probability that the maximum number of merging operations performed next by the simulator will be either two or three.

The technique maintains the properties of the SFQ scheduler by merging requests in increasing number of start tags. Furthermore, only requests of the same class are merged. As a result, the algorithm aborts in cases where the queued request with the minimum start tag is of a different class as the already merged requests. Once a merged job has received service and exits the model, each of the merged requests counts as a separate completion. Since each of the requests sharing a service station has an individual service time, we approximate the aggregate service requirement of merged requests with the mean of the individual request service times.

3.6.3.2 Merge Value Estimation

The challenge is to calibrate our model without detailed knowledge of the VMM internals. To estimate the extent of merging operations in this black box view of the VMM we have developed an iterative search technique. Our technique controls the mean number of requests in simulation experiments, i.e. the mean queue length, through the ω parameter and terminates once the mean queue length seen in simulation closely matches an inferred *expected* queue length, as follows.

Inference of Mean Expected Queue Length. The first step of merge value estimation is to

Table 3.1: Summary of Input and Output Parameters for the Homogeneous Model.

Input	$A(c_k^i)$	Arrival time trace measured from VMs running in isolation. Traces are further processed to account for environment specific VMM request size thresholds.
	$D(c_k^i)$	Service times sampled from a MAP, fitted from measured traces of VMs running in isolation.
	l_{max}	Maximum request size threshold of VMM environment.
	H	Number of servers based on VMM to LUN issue queue length configuration.
	ω	Model calibration parameter estimated in iterative search algorithm.
Output	$C(c_k^i)$	Predicted request response times in consolidation.

Table 3.2: Measurements and Expected Mean Number of Requests N in the System in Isolation (Iso) and Consolidation Scenarios with Two VMs (Con 2) and Three VMs (Con 3).

Workload	*Iso*	*Con 2*		*Con 3*	
	N_{meas}^{iso}	N_{meas}^{2}	N_{exp}^{2}	N_{meas}^{3}	N_{exp}^{3}
PM-1	11.9	27.5	23.8	43.7	35.7
PM-2	14.1	30.2	28.2	44.6	42.3
FFSB-1_S	4.6	8.5	9.2	12.8	13.8
FFSB-1_R	4.8	8.5	9.6	13.3	14.4
FFSB-2_S	3.5	6.4	7.0	9.0	10.5
FFSB-2_R	4.0	7.5	8.0	10.0	12.0

infer the expected mean queue length in the system for the consolidation scenario under study. We infer the expected mean queue length for a consolidation scenario with K classes based on the assumption that the mean number of requests in the system grows linearly when moving from isolation to consolidation:

$$N_{exp}^{K} = \sum_{i=1}^{K} N_{meas}^{iso}, \tag{3.7}$$

where K is the total number of request classes considered in the simulation model, N_{exp}^{K} is the expected mean queue length in simulation, and N_{meas}^{iso} is a measurement of the mean queue length obtained in isolation benchmark experiments. The queue length parameters we consider include requests that have been issued by the VM operating system and are either queued at the VMM or in service, i.e. pending, at the LUN storage device.

To validate this linear assumption Table 3.2 shows measurements of the mean number of requests from benchmark experiments in our reference system. We present measurements for a

number of workload configurations averaged over 15 benchmark runs. For detailed information on the considered workloads see Section 3.7.1. Results indicate that the linear assumption is a good approximation of system behavior. We expect our assumption to hold as long as the aggregate mean number of requests outstanding from VMs, i.e. requests issued and not yet completed, does not exceed the number of available connections from the VMM to the LUN.

Iterative Search. The expected queue length approximation is an input parameter for an iterative search technique, which we propose to estimate the merge value parameter for the simulation model. As shown in Algorithm 2, the search is further parameterized with a configurable initialization point and a maximum relative error value, Δ_{max}, that serves as a search termination condition. Each search iteration begins with a number of simulator runs that incorporate merging operations according to the current value of ω. Every simulator run is parameterized with a random combination of interarrival time traces drawn from a trace repository, depending on the number and type of considered request classes *k*. At the end of each search iteration we compute the corrected mean queue length in simulations, N'_{sim}, with

$$N'_{sim} = \frac{N_{sim}}{\omega},\ \omega \geq 1 \tag{3.8}$$

where N_{sim} is the mean queue length over all simulation runs and N'_{sim} represents the effective queue length after the merging transformation with ω. The effective queue length in simulation is then used as input parameter for the function get_merge_error, which computes the relative error Δ_ω produced by the current ω estimate according to the error function

$$\Delta_\omega = \left| \frac{N^K_{exp} - N'_{sim}}{N^K_{exp}} \right|, \tag{3.9}$$

where N^K_{exp} is the inferred expected queue length computed according to (3.7) from isolation experiments. The search terminates if the corrected queue length is accurate within 5% of N^K_{exp}. In cases where the estimation error is outside this range, we control search direction and ω values on the basis of a binary search. Let $\omega = g(N'_{sim})$ be the merge value used in simulation to obtain N'_{sim}. If N'_{sim} is smaller than N^K_{exp}, we decrease ω in order to increase the mean number of requests in simulation. In cases where previous iterations have produced a $N'_{sim,old}$, such that $\{N'_{sim,old} > N^K_{exp} > N'_{sim}\}$ the merge value for the next iteration is determined by

$$\omega = g(N'_{sim}) - \frac{g(N_{sim})' - g(N'_{sim,old})}{2}, \tag{3.10}$$

which is half the distance between ω values used to obtain N'_{sim} and $N'_{sim,old}$. In cases where

Algorithm 2 Iterative Estimation of Merge Value

```
ω ← merge value initialization point
N_exp ← inferred expected queue length
Δ_max ← 0.05
flag ← 0
while flag != 1 do
  #run configurable amount of simulator iterations
  for i = 1 to max_simulator_iterations do
    for k = 1 to K do
      draw random arrival trace from repository
    end for
    simulate(ω)
  end for
  #search merge value ω
  N_sim ← mean queue length over simulator iterations
  N'_sim ← (N_sim/ω)
  Δ_ω ← get_merge_error(N'_sim)
  if Δ_ω ≤ Δ_max then
    flag ← 1
  else if N'_sim < N_exp then
    ω ← decrease
  else if N'_sim > N_exp then
    ω ← increase
  end if
end while
```

no such $N'_{sim,old}$ exists, we decrease ω by a configurable step size parameter. The inverse of the above applies for the opposite search direction. Table 3.1 offers a summary of the input and output parameters for the Homogeneous model.

3.7 Validation Experiments: Homogeneous Model

In this section we validate the proposed methodology with data obtained in a number of benchmark experiments from our reference system. We first present the benchmark applications considered and their measured workload characteristics. We later compare our prediction results with measured results from consolidation scenarios and predictions from an analytical product form queueing model.

3.7.1 Workload Generation

We consider a number of different workload types, which are submitted from within VMs running in isolation, as well as in consolidated scenarios. The workloads consist of varying

Table 3.3: Workload Configurations.

	Parameter	Conf-1	Conf-2
PM	Size low bound	500 byte	9.77 kB
	Size high bound	9.77 kB	19.53 kB
	Size read	512 byte	2 kB
	Size write	512 byte	2 kB
FFSB	Size Read	4 kB	32 kB
	Size Write	4 kB	32 kB

configurations of two benchmarks with quite distinct characteristics. The Flexible File System Benchmark (FFSB) [3] is a multi-threaded benchmark comprising large file operations. Conversely, Postmark (PM) [9, 119] is a synthetic single-threaded workload comprising small file and metadata-intensive operations designed to emulate I/O patterns of Internet applications such as e-mail and e-commerce. We specifically choose these two workload types to validate our model since the different file size distributions should result in distinct quantities of VMM I/O request splitting operations.

In order to obtain a system steady state, benchmarks for each configuration are submitted over a period of 75 minutes in five minute intervals. While the FFSB application can be parameterized to run for a specified time frame, PM does not provide such an option. In cases where a PM run needs less than five minutes to complete we restart it, but only collect monitoring data within the five minute time window that captures a run. In total we run 720 benchmarks and collect roughly 87 million measurement samples for our analysis. See Appendix C for more detailed statistics of the conducted benchmark experiments.

PM. File servers running Internet applications typically work with a large number of short-lived, relatively small files (1kB-100kB). PM was designed to emulate scenarios of this domain, where at any given time files are being rapidly created, deleted, read, or written, at random locations on the storage device. At the beginning of each benchmark run an initial pool of random text files is created, with sizes ranging in configurable bounds. Then a specified number of create/delete and read/append operations occur. Finally, upon completion of all operations the set of files is deleted.

We have considered two configurations of a PM workload in our investigation, denoted PM-1 and PM-2. The workloads differ in the file sizes of the initial file set, as well as the sizes of read and write requests (see Table 3.3). The "size" parameters specify how much data is read or written to files at a time. When using PM-2 the benchmark creates a file set spanning a large size range and performs reads/writes of increased sizes. In both PM configurations read/append, as well as create/delete operations are equally likely to occur.

FFSB. Similar to PM we have defined two FFSB configurations, but this benchmark addi-

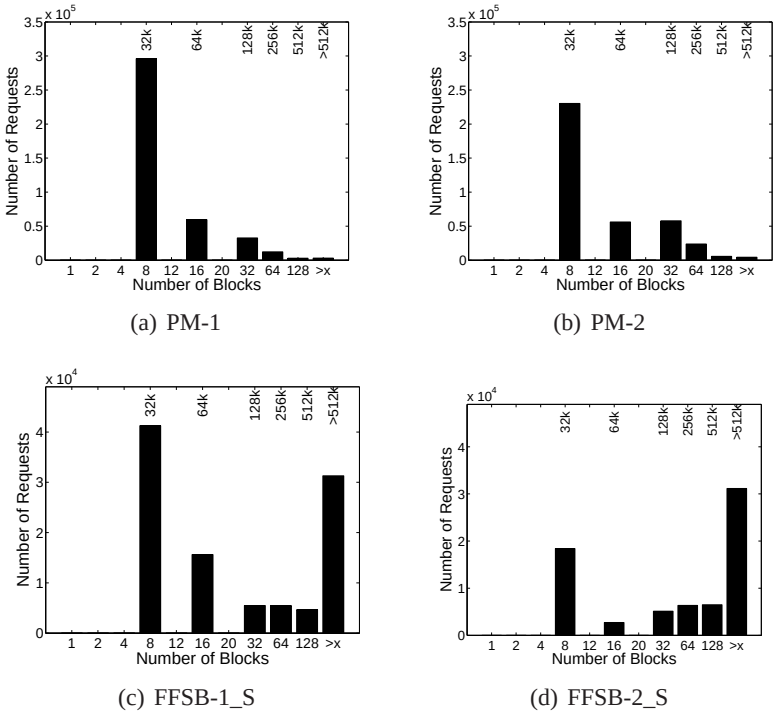
tionally supports the execution of *sequential*, as well as *randomized* reads/writes. The response times of sequential and random requests can significantly differ, since sequential requests are most directly affected by the transfer speed of the disk drive, while random requests are most directly affected by disk seek time [21]. Sequential read requests can also take advantage of *read-aheads*, which assume that an application reading from disk block n will next ask to read from the subsequent disk blocks. The read-ahead mechanism decreases seek times by caching the following disk blocks $n+1, n+2, etc.$, into memory, but is turned off once the system detects a non-sequential file access. Considering the sequential and randomized options essentially leaves us with four distinct FFSB workloads for our study, denoted as FFSB-1_S, FFSB-1_R, FFSB-2_S, and FFSB-2_R.

At the beginning of each benchmark run FFSB dedicates a single thread to create an initial file set for the subsequent operations. In our specific configuration the application creates a structure of 400 files and 10 directories. Even though both configurations are initialized with the same total number of files, the distribution of file sizes is different. FFSB allows us to specify the quantity of certain file sizes, as well as the probability of I/O operations via relative weights. The file sizes in FFSB-1_S/R and FFSB-2_S/R range in $[4\text{kB}, 32\text{MB}]$ and $[32\text{kB}, 32\text{MB}]$, respectively, with the highest request frequencies for the smaller sizes. We define a single threadgroup consisting of 100 threads and set the think time parameter to zero. Read and write I/O operations are configured with the same weight, but different sizes, as shown in Table 3.3.

See Appendix D for the benchmark configuration files used throughout our PM and FFSB benchmark experiments.

3.7.2 Workload Characterization

In this section we present how workloads submitted by the benchmark application are translated into logical block requests by the VM operating system. The disk I/O scheduler of the VM operating system aggregates and reorders queued requests. As a result the total number and sizes of disk requests issued by the VM operating system to the VMM can significantly differ from the total number and sizes of requests submitted by the application to the VM operating system. We first illustrate the request size distribution and the interarrival time distribution for requests submitted by a VM to the VMM. In particular, we show the effect of workload consolidation on interarrival times of requests issued by a VM. Finally, we also present service time statistics for all workload configurations considered.



(a) PM-1 (b) PM-2

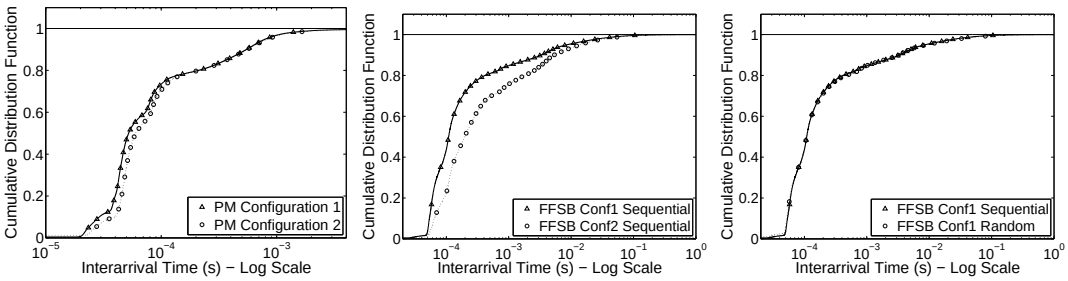
(c) FFSB-1_S (d) FFSB-2_S

Figure 3.8: Measured Distribution of Request Sizes for PM and FFSB Workload Configurations.

3.7.2.1 Request Size

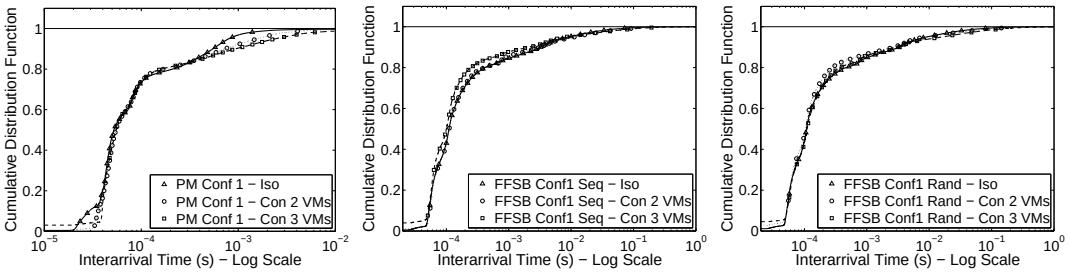
To illustrate the different size characteristics of the workload configurations considered, Figure 3.8 shows measurements of VM disk request size distributions of sample benchmark experiments. For ease of presentation we have grouped the data into bins. The sizes of the bins are chosen on the basis of a related workload characterization study [34]. As shown in Figure 3.8 (a) the VM kernel aggregates the majority of requests submitted by PM-1 into sizes larger than four and less or equal to eight blocks. Since the standard file system block size on our reference system is 4kB, a request for eight blocks has a size of 32kB. Figure 3.8 (b) reflects the main difference between the two PM workload configurations. PM-2 submits requests of larger sizes. This results in a lower frequency of 32kB (eight blocks) request sizes and additional requests for block sizes greater than eight. A common attribute of the two PM workloads is that neither workload causes the VM scheduler to issue a significant number of requests with blocks sizes greater than 128.

Figures 3.8 (c) and (d) show request size distributions of FFSB workloads and reveal that workload characteristics are quite different compared to PM workloads. Similar to PM the VM scheduler aggregates a large number of the FFSB workload to requests of size 32kB (eight blocks). However, the total number of requests is significantly lower and the proportion of requests with large sizes is significantly higher than in the case of PM. Evaluating the main



(a) PM (b) FFSB-1_S and FFSB-2_S (c) FFSB-1_S and FFSB-1_R

Figure 3.9: Distribution of Arrival Processes of PM and FFSB Workloads in Isolation.



(a) PM-1 (b) FFSB-1_S (c) FFSB-1_R

Figure 3.10: Impact of Workload Consolidation on Arrival Processes of PM and FFSB Workloads.

differences between FFSB-1_S and FFSB-2_S, the increased file sizes and frequency of large file operations of FFSB-2_S allow the scheduler to translate FFSB-2_S workloads into fewer requests of larger sizes as seen in Figure 3.8 (d). The total number of requests compared to FFSB-1_S decreases, while the frequency of requests of block size 64 and above increases. For both FFSB workloads large proportions of requests are merged to block sizes > 128, which corresponds to request sizes > 512kB.

3.7.2.2 Interarrival Times

The significantly different characteristics of the considered workload configurations are also reflected in the distributions of request interarrival times. Figure 3.9 (a) shows the interarrival time distributions of requests to the VMM, as recorded when submitting the PM workloads. While both distributions have a similar shape, there is a higher probability for longer interarrival times in PM-2. We computed the mean interarrival times for PM-1 and PM-2 as 0.72ms and 0.81ms, respectively, as shown in Table 3.4.

Figure 3.9 (b) illustrates the arrival distributions for both FFSB configurations when performing sequential read/write operations. Compared to the PM workloads, the interarrival times are considerably longer. We observe a significantly higher probability for longer interarrival times

Table 3.4: Mean in ms, Standard Deviation (std), and Coefficient of Variation (cv) Statistics for Request Interarrival and Service Times.

Workload	*Interarrival Times*			*Service Times*		
	mean	std	cv	mean	std	cv
PM-1	0.72	15.4	21.0	8.61	43.4	5.0
PM-2	0.81	13.4	16.4	11.5	58.8	5.1
FFSB-1_S	3.07	28.5	9.2	13.9	46.8	3.4
FFSB-1_R	3.24	29.6	9.1	15.6	49.1	3.1
FFSB-2_S	4.36	43.4	9.9	15.2	52.4	3.4
FFSB-2_R	3.54	38.2	10.8	14.1	50.1	3.6

Table 3.5: Mean in ms, Standard Deviation (std), and Coefficient of Variation (cv) Statistics for Request Batch Sizes.

Workload	*Batch Size*		
	mean	std	cv
PM-1	4.04	2.52	0.62
PM-2	4.14	3.44	0.83
FFSB-1_S	4.98	6.39	1.28
FFSB-1_R	5.27	6.77	1.29
FFSB-2_S	5.26	6.33	1.20
FFSB-2_R	5.30	6.94	1.31

for FFSB-2_S with a 95th percentile of 13.6ms. FFSB-1_S has a shorter mean interarrival time across the whole range of the distribution with the 95th percentile of interarrival times being 8.5ms or less. The distribution of arrivals does not seem to be affected by randomizing disk access, as Figure 3.9 (c) indicates.

Since our model uses information recorded during isolation benchmark experiments only, we are especially interested in quantifying the impact of workload consolidation on request interarrival times. Figure 3.10 shows the interarrival time distributions of disk requests submitted from VMs when running in isolation (Iso) compared to consolidation scenarios with additional VMs submitting an identical workload in the background. Interestingly none of the arrival distributions from the VM to the VMM displays a large deviation from their corresponding isolation distributions when the workload on the server is doubled (Con 2) or tripled (Con 3). We take this as a clear indication that queueing of disk I/O requests due to resource contention takes place at the VMM layer, rather than at the VMs themselves.

3.7.2.3 Service Times

Our methodology approximates the service requirement of disk requests with measured response times in isolation scenarios as described in Section 3.6.2.2. In isolation the utilization

levels at the VMM are likely to be low and thus requests may not incur queuing. Table 3.4 shows mean service time statistics for all workload configurations averaged over all VMs and benchmark runs. The service requirements for FFSB workloads are higher than for PM. This is probably due to the larger request sizes of FFSB which entail increased lengths of read/write operations at the disk drive. Interestingly, randomizing workload patterns does not automatically lead to higher service times. FFSB-1_R has a larger service time than FFSB-1_S, while it is the opposite for FFSB-2_R and FFSB-2_S.

3.7.2.4 Batch Sizes

The batched submission of I/O requests results in bursty arrival patterns of disk workloads [169]. Burstiness has been identified to impact system performance [67, 147] and complicates modeling efforts [146]. We include an analysis of batch sizes for the considered workloads and structure this section in two parts. First we use an example to explain our custom method to locate batches in a time series before reporting batch size measurements.

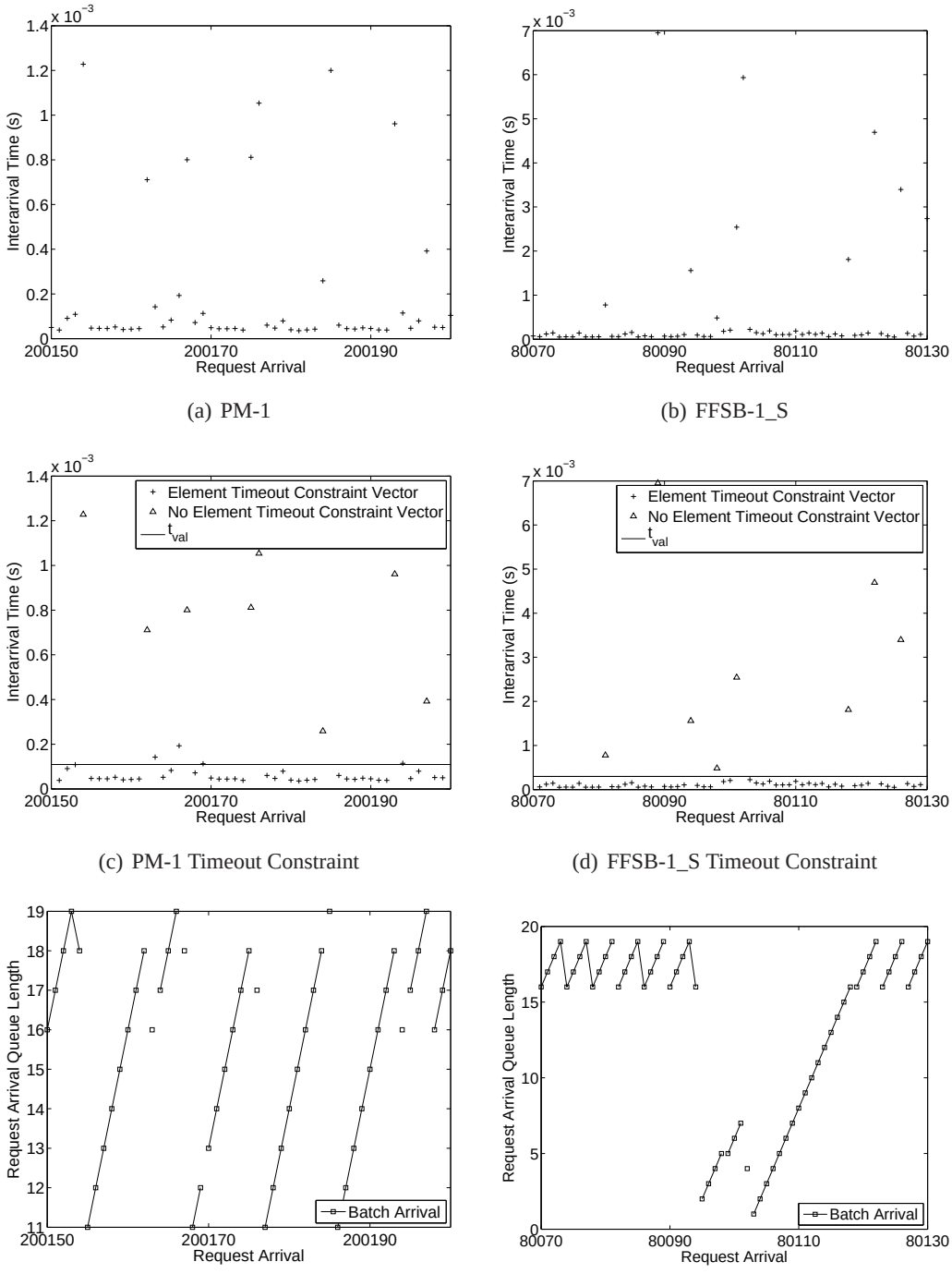
Figures 3.11 (a) and (b) show example intervals of measured interarrival times to illustrate how the VM schedulers submit PM and FFSB requests to the disk. The samples convey a varying number of consecutive requests submitted within a short time interval, i.e. with small interarrival times, followed by a single arrival with an increased interarrival time. We consider this observation in our analysis and define the elements of a batch as

$$\forall c_k^i \in B_n : \text{diff}\{A(c_k^{i+1}), A(c_k^i)\} \leq t_{val}, \tag{3.11}$$

where batch B_n holds all arrivals c_k^i that have interarrival times ranging within a timeout constraint t_{val}. In order to approximate t_{val} we filter the interarrival time series of a representative run from the increased values that mark interfaces between consecutive batches. The timeout constraint vector t is defined as

$$\forall A(c_k^i) \in t : \frac{A(c_k^i)}{A(c_k^{i-1})} \leq h, \tag{3.12}$$

where h is an upper bound on the ratio between the interarrival times of two consecutive requests. We approximate the upper bound as $h = 3$, i.e. interarrival times that are more than three times larger compared to the predecessor are not included in t. The timeout constraint value, t_{val}, is set to the 95th percentile of t in order to exclude possible outliers. Figures 3.11 (c) and (d) illustrate how elements of t are filtered from the previous example series of interarrival times using our technique. Finally, we pass t_{val} as an input to Equation (3.11) and determine batch locations and batch sizes in the measured arrival trace. Figures 3.11 (e) and (f) complete our example and show batch arrivals identified by our technique.



(a) PM-1

(b) FFSB-1_S

(c) PM-1 Timeout Constraint

(d) FFSB-1_S Timeout Constraint

(e) PM-1 Detected Batches

(f) FFSB-1_S Detected Batches

Figure 3.11: Example of Custom Technique for Workload Batch Arrival Analysis.

The statistics reported in Table 3.5 convey that mean batch sizes are similar for all workloads. However, the variation in batch sizes is almost twice as large in FFSB workloads compared to PM, with coefficients of variation ranging in $[1.20, 1.31]$ and $[0.62, 0.83]$ for FFSB and PM, respectively. For two reasons we are especially interested to learn how response times of individual requests relate to the *joint* response times of batch arrivals:

1. Our methodology uses response times recorded in low utilization benchmark experiments to approximate service requirements as described in Section 3.6.2.2.
2. The model calibration technique in Section 3.6.3 can merge a configurable number of consecutive arrivals into a single unit and requires approximation of service times of merged requests.

We introduce the term *batch window size* to specify the response time of a batch unit and define this quantity as

$$W(B_n) = \max(E(c_k^i)) - \min(A(c_k^i)), \quad \forall c_k^i \in B_n \tag{3.13}$$

where $W(B_n)$ is the window size of batch B_n, $E(c_k^i)$ is the departure time of request c_k^i, and $A(c_k^i)$ is the arrival time of request c_k^i. Furthermore, we denote the mean response time of requests that are part of a batch unit as

$$C(B_n) = \frac{1}{|B_n|} \sum C(c_k^i), \quad \forall c_k^i \in B_n \tag{3.14}$$

where $|B_n|$ is the size of batch B_n. Table 3.6 shows mean and median statistics for $W(B_n)$ and $C(B_n)$. We report these metrics during varying utilization levels and compare the relative difference as

$$\Delta_B = \left| \frac{W(B_n) - C(B_n)}{W(B_n)} \right|. \tag{3.15}$$

The results from isolation scenarios show considerably lower deltas for the median compared to the sample mean indicating that the latter may be affected by outliers. Delta values of the median are generally low ranging in $[0.09, 0.20]$. In the higher utilized case of Con2 the relative differences between batch windows and mean batch request response times decrease: Delta values range in $[0.23, 0.36]$ and $[0.05, 0.12]$ for mean and median, respectively. Our analysis of low utilization Iso cases shows that the time taken to serve a batch arrival, i.e. the service requirement of a batch, may be approximated based on the service times of batch request elements. It is interesting to note that the accuracy of this approximation increases in higher utilizations, since the modeling of consolidation scenarios is the focus of this study.

Table 3.6: Mean and Median Statistics in ms for Batch Window Sizes $W(B_n)$, Batch Request Response Times $C(B_n)$, and the Relative Error Δ_B

Workload		*Iso*			*Con 2 (VM 1)*		
		$W(B_n)$	$C(B_n)$	Δ_B	$W(B_n)$	$C(B_n)$	Δ_B
PM-1	*mean*	19.8	9.30	0.53	37.5	27.3	0.27
	median	2.22	1.87	0.13	4.39	3.94	0.07
PM-2	*mean*	25.8	12.3	0.52	44.8	28.8	0.36
	median	2.39	2.02	0.14	4.23	3.76	0.08
FFSB-1_S	*mean*	21.7	15.2	0.30	28.6	22.0	0.23
	median	10.7	7.49	0.10	13.2	10.3	0.06
FFSB-1_R	*mean*	28.0	19.1	0.32	34.6	25.2	0.27
	median	12.0	8.67	0.09	11.7	9.48	0.05
FFSB-2_S	*mean*	22.4	14.5	0.35	29.4	21.4	0.27
	median	14.8	8.60	0.19	20.2	13.9	0.11
FFSB-2_R	*mean*	25.9	17.1	0.34	29.8	21.8	0.27
	median	14.5	8.42	0.20	19.4	13.4	0.12

3.7.3 Model Validation

In this section we study the accuracy of our proposed methodology to predict the mean response time of disk requests when consolidating storage workloads. For validation we conduct a series of benchmark experiments on our reference system using the previously introduced workload configurations. Workloads are submitted from within VMs. We consider scenarios with up to three VMs, denoted VM 1, VM 2, and VM 3, where we consolidate homogeneous workloads, i.e., workloads of the same type. We specifically choose this number of VMs as it resembles a realistic situation for the consolidation of disk I/O intensive applications and the storage contention scenario in our reference system [24]. Each benchmark experiment consists of 15 individual 5 min runs and results are reported as means over all runs. We first show measurement results before comparing the prediction accuracy of our simulation model to a product form analytical solution, as well as a baseline model without calibration heuristics.

3.7.3.1 Measurement Results

We present measurements that illustrate the impact of workload consolidation on disk request response times. Disk response times are an important metric since they may directly affect the performance of end users of applications, e.g. in cases where processes block until completion of a read request. All results are based on in-VM measurements obtained with the *blktrace* tool as described in Section 3.5.2.

Table 3.7 shows that workload consolidation leads to an increase in disk I/O response times across all workload configurations. The PM workloads suffer a higher performance degradation

Table 3.7: Mean Response Time Measurements of Disk I/O Requests in ms for VMs in Isolation (Iso), Confidence Interval Width of Isolation Measurements (CI_{iso}), and Mean Response Time Measurements of Consolidation Scenarios with Two VMs (Con 2) and Three VMs (Con 3).

Workload	*VM 1*				*VM 2*				*VM 3*		
	Iso	CI_{iso}	Con 2	Con 3	Iso	CI_{iso}	Con 2	Con 3	Iso	CI_{iso}	Con 3
PM-1	9.45	3.7	28.3	47.7	8.49	3.1	25.4	48.2	7.9	4.5	37.8
PM-2	11.5	5.0	31.1	53.3	11.9	3.9	30.8	56.0	11.0	3.6	43.9
FFSB-1_S	14.8	27.0	18.4	24.9	13.5	6.0	16.8	24.5	13.3	8.3	25.2
FFSB-1_R	16.4	5.6	21.5	29.8	15.2	6.8	20.1	25.6	15.3	6.7	28.2
FFSB-2_S	15.4	5.9	19.9	24.9	15.2	5.9	20.4	22.3	15.1	5.9	23.6
FFSB-2_R	14.6	5.1	19.6	23.5	13.7	5.8	19.4	22.9	13.8	4.8	23.3

than FFSB, with an increase ranging in approximately $[158\%, 199\%]$ and $[299\%, 468\%]$ in the two and three VM consolidation scenarios, respectively. In case of the FFSB workloads this increase is less severe, but still ranges approximately in $[26\%, 41\%]$ and $[47\%, 81\%]$ over all VMs and configurations. Furthermore, there is no clear trend showing that response times of random FFSB workloads increase to a larger degree than sequential FFSB workloads when consolidating. Interestingly, VMs seem to have slightly different I/O characteristics even for identical workload configurations. For example, in a three VM consolidation scenario PM-2 requests submitted from VM 2 have a mean response time of 56ms, while the mean response time of PM-2 requests submitted from VM 3 is 43.9ms. Such discrepancies may be explained due to the spatial locality of VM disk images on the LUN. We indicate the stability of our monitoring using a 95% confidence interval and report the confidence interval width CI_{iso} as the percentage of the mean measured response time in isolation over all benchmark runs.

3.7.3.2 Prediction Accuracy

In this section we present predictions of mean disk request response times obtained using the proposed methodology. We validate model predictions against system measurements. We then compare the quality of our predictions to an open product form solution, which has previously been successfully applied in environments with shared resources [52]. Furthermore, our model is evaluated against a baseline model without splitting and merging heuristics to show the effectiveness of the proposed model calibration techniques. Response time estimates for the product form model are determined analytically as

$$C_{est,k} = \frac{D_k}{1 - \sum_{t=1}^{K} \frac{\lambda_t}{n} \times D_t}, \tag{3.16}$$

where $C_{est,k}$ is a response time estimate for class k requests, D is the mean service time, λ the mean arrival rate, K the number of request classes, and $\{n = 32\}$ the number of servers in the model.

Table 3.8: Confidence Interval Width of Simulation Results (CI_β), as well as Mean Relative Errors of Response Time Predictions for Simulation (Δ_ω), Base (Δ_b), and Product Form (Δ_p) Model.

Workload	*Con 2*					*Con 3*				
	CI_β	Δ_ω	CI_β	Δ_b	Δ_p	CI_β	Δ_ω	CI_β	Δ_b	Δ_p
PM-1	11.6	0.13	7.06	6.58	0.46	17.9	0.09	20.7	427.6	18.4
PM-2	8.9	0.08	11.3	8.84	2.26	10.3	0.06	7.56	984.6	64.9
FFSB-1_S	2.3	0.03	3.14	0.02	0.12	12.3	0.02	6.83	0.04	0.06
FFSB-1_R	4.2	0.03	3.06	0.04	0.09	27.7	0.30	10.1	0.23	0.03
FFSB-2_S	6.6	0.09	0.69	0.20	0.03	19.4	0.04	1.19	0.31	0.05
FFSB-2_R	1.8	0.16	1.64	0.20	0.03	5.05	0.15	2.9	0.23	0.04

Table 3.9: Comparison of Merging and Splitting Operations Performed by the Simulation Model.

Workload		*Con 2*	*Con 3*
	Ψ	ω	ω
PM-1	1.01	2.55	4.7
PM-2	1.02	2.43	4.35
FFSB-1_S	1.56	2.0	2.9
FFSB-2_S	1.83	2.2	3.075
FFSB-1_R	1.54	2.0	3.9
FFSB-2_R	1.64	1.9	2.575

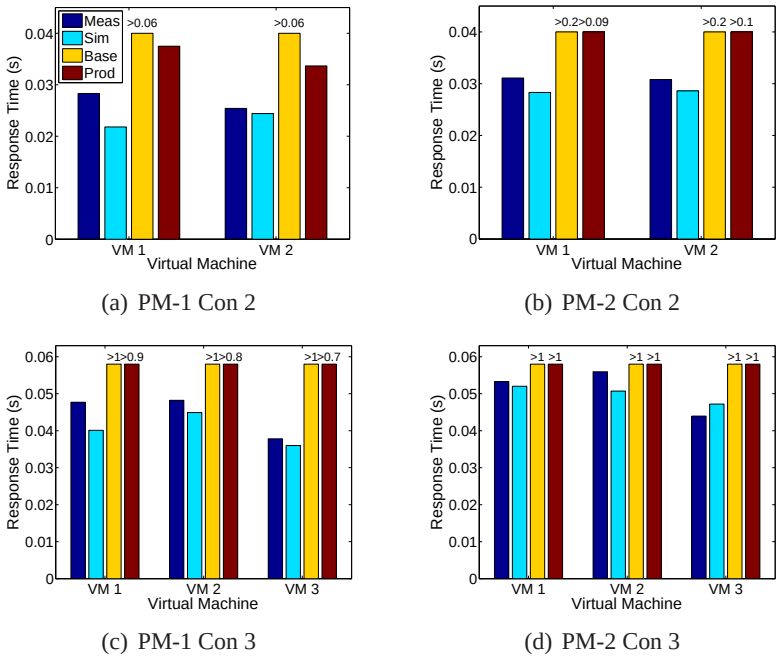
Predictions of our simulation model are averaged over multiple simulation runs, with the number of runs ≥ 50 and ≥ 100 for PM and FFSB simulations, respectively. We indicate the reliability of the estimate using a 95% confidence interval and report the confidence interval width CI_β as the percentage of the mean, averaged over all classes. Furthermore, the model incorporates some specific characteristics of our reference system. The VMM splits incoming traffic above a size threshold of 512kB (128 blocks), which we consider in the parameterization of the model as described in Section 3.6.2.1. The quantity of splitting operations is reported as

$$\Psi = \frac{J^I_{split}}{J^I}, \tag{3.17}$$

where J^I is the total number of request arrivals before our splitting step, J^I_{split} the total number split of arrivals, and Ψ the splitting ratio. Prediction accuracy is evaluated by the error function

$$\Delta = \sum_{k=1}^{K} \frac{1}{K} \left| \frac{C_{meas,k} - C_{est,k}}{C_{meas,k}} \right|, \tag{3.18}$$

which is the mean relative error over all k classes of the estimated response time $C_{est,k}$ with respect to the measured value $C_{meas,k}$.



(a) PM-1 Con 2

(b) PM-2 Con 2

(c) PM-1 Con 3

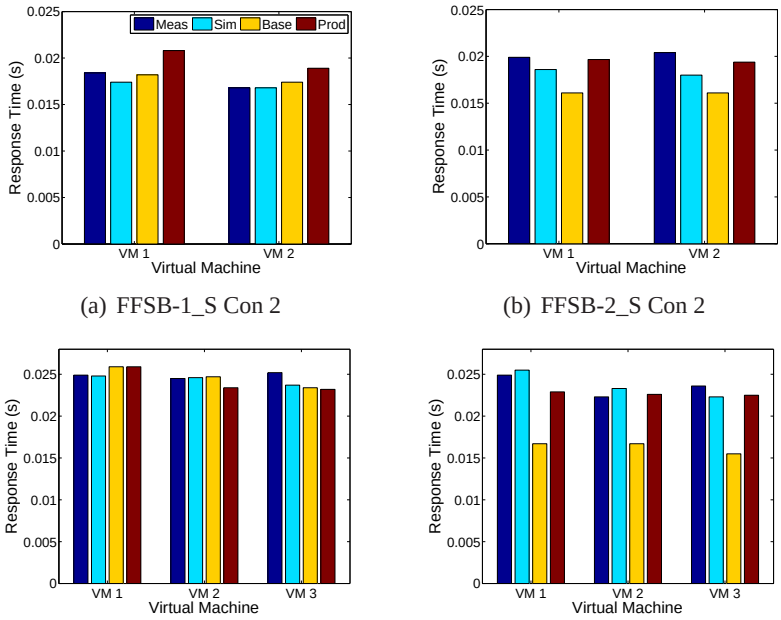
(d) PM-2 Con 3

Figure 3.12: Comparison of Response Times from Measurement (Meas), Simulation (Sim), Simulation Base-Line (Base), and Product-Form (Prod) Model for Two and Three VMs Consolidation Scenarios With Postmark Workloads. Legend Shown in Figure (a) is Identical for Figures (b), (c), and (d).

PM. The simulation model delivers accurate prediction results for PM-1 and PM-2 in both consolidation scenarios, as shown in Table 3.8. In light of the extent to which response times of these workloads increase in consolidation, the quality of results is highly satisfactory. Conversely, the product form model delivers larger errors and is not competitive except for the case of PM-1 in Con 2. The effectiveness of the model calibration technique is further highlighted by the large errors of the baseline model: Table 3.9 shows that we estimate large ω values especially for the Con 3 consolidation cases of the PM workload.

As we have shown in Figures 3.8 (a) and (b) the numbers of requests larger than 512kB are very small for PM workloads, thus splitting operations are negligible. Figure 3.12 shows that predictions of the simulation model underestimate the measured response times, except for the case of VM3 in PM-2 Con 3. We reason this might be due to the necessary approximation of service times for merged requests, where we estimate the aggregate service requirement with the mean. Even though it is likely that the storage device serves merged requests asynchronously, this might be an optimistic approximation. Results for product form and baseline models are both grossly overestimating the measured response times.

FFSB. Simulation and product form models deliver good results when predicting response times of sequential FFSB workload configurations. The simulation model performs especially



(a) FFSB-1_S Con 2

(b) FFSB-2_S Con 2

(c) FFSB-1_S Con 3

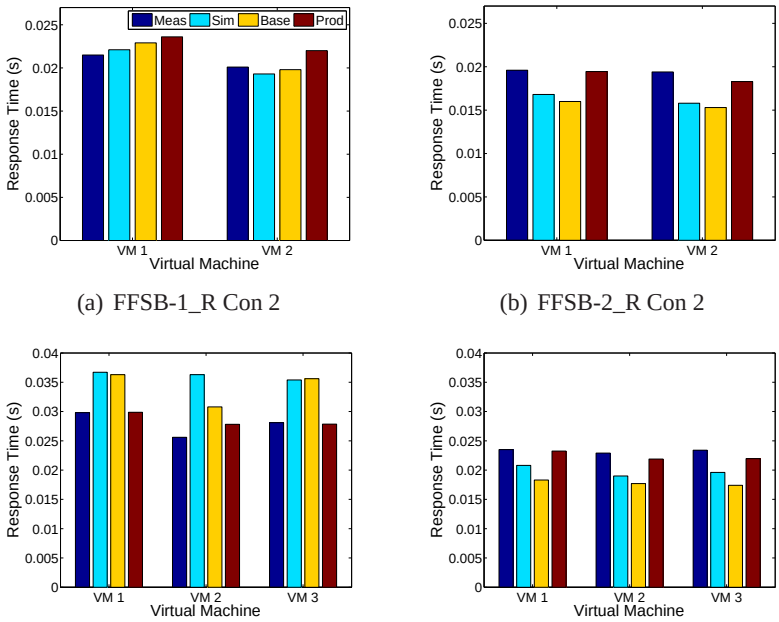
(d) FFSB-2_S Con 3

Figure 3.13: Comparison of Response Times from Measurement (Meas), Simulation (Sim), Simulation Base-Line (Base), and Product-Form (Prod) Model for Two and Three VMs Consolidation Scenarios With FFSB Sequential Workloads. Legend Shown in Figure (a) is Identical for Figures (b), (c), and (d).

well for the cases of FFSB-1_S. Interestingly, the product form model works significantly better than for the PM workloads. Our technique performs VMM level splitting operations on the arrival trace, as well as merging, as shown Table 3.9. We have particularly found the splitting behavior difficult to model, as it needs to maintain temporal dependence in the measured arrival trace. Furthermore, one needs to rejoin split requests and make additional approximations on the aggregate response time. For the case of Con 2 splitting and merging operations in our model almost compensate each other, explaining the competitive results of the baseline model.

Figure 3.13 shows that our model predicts optimistic response time values, except for the case of FFSB-2_S Con 3. The baseline model outputs larger errors for the FFSB-2 workload configurations. Table 3.9 reports increased split ratios and ω values for FFSB-2_S thus further confirming the effectiveness of our proposed methodology.

Table 3.8 conveys that the product form model works well for the workloads with random disk access patterns. Errors of our model are marginally larger, but except for the difficult case of FFSB-1_R range in $[0.03, 0.16]$. With FFSB-1_R we also find the only case of our study where the baseline model without calibration delivers noticeably better results than the heuristically calibrated one. Figure 3.14 further shows that most of our modeling results are optimistic. Only in the case of FFSB-1_R Con 3 the simulation and baseline models predict larger response times compared to measurement.



(a) FFSB-1_R Con 2

(b) FFSB-2_R Con 2

(c) FFSB-1_R Con 3

(d) FFSB-2_R Con 3

Figure 3.14: Comparison of Response Times from Measurement (Meas), Simulation (Sim), Simulation Base-Line (Base), and Product-Form (Prod) Model for Two and Three VMs Consolidation Scenarios With FFSB Random Workloads. Legend Shown in Figure (a) is Identical for Figures (b), (c) and (d).

3.8 Summary and Conclusions

Conclusions. We presented a trace-driven simulation model that can predict response times of disk I/O requests when consolidating homogeneous workloads on to a shared storage device. Our contributions are twofold. Firstly, we parameterize the model with in-VM measurement data only instead of instrumenting the VMM. Secondly, we introduce techniques that incorporate VMM level optimization operations such as I/O request splitting and calibrate the model with an iterative search technique that merges queued requests.

The proposed methodology is validated against system measurements using a range of synthetic workloads and produces better results than an established product form solution for certain homogeneous workload types. Our findings indicate that prediction quality of disk request response times in homogeneous consolidation scenarios can be increased with enhanced models, which can account for splitting operations of the VMM disk scheduler and are heuristically calibrated.

Limitations. The approach is limited to homogeneous workload consolidations. However, the range of application for such models significantly widens if they are also able to accurately predict response times in heterogeneous consolidation scenarios. Model calibration with multiple workload types is more difficult since each type requires individual ω merge values

and thus complicates the merge value estimation step. We leverage our findings from modeling homogeneous workload consolidations and tackle this additional complexity with a model refinement as described in Chapter 4.

4 I/O Performance Prediction in Heterogeneous Consolidated Environments

4.1 Introduction

In this chapter we propose models for predicting the performance of heterogeneous storage workload consolidations, i.e. disk workloads of different types. Such models are more powerful and realistic than homogeneous ones, because they support system administrators in finding the workload mixes with the least interference effects and can be especially important in Cloud environments that employ virtualization technologies. In virtualized data centers the concurrent shared use of a storage device by several Virtual Machines (VMs) submitting different disk workloads may lead to significant performance degradation [124]. Estimates of disk I/O contention for a given VM placement configuration can support management and consolidation decisions in such systems.

In particular, we introduce simple models for I/O performance prediction in consolidated environments where multiple VMs can share access to a remote storage server. Our methodology, summarized in the previously introduced Figure 3.1, requires first a study of VMs when they run in isolation on a virtualized server. Based on collected measurements we propose two types of models to forecast the impact of consolidation on I/O performance. Specifically, for each VM of interest we collect traces of arrival times, estimated service times, and arrival queue lengths for I/O requests.

We develop two modeling techniques that embody this approach. We start by developing what we denote as the Linear Estimation Formulas to forecast mean read/write mixes, throughputs, and response times in consolidation. Additionally, we introduce a simulation model for predicting response time distributions. Such a capability is particularly useful for cloud operators interested in providing Service Level Agreements (SLAs) to customers.

Our models support distinction between read and write requests by decomposing the workload according to request type. Read and write requests may affect system performance in very different ways. Ganger and Patt [91] introduce three classes of requests based on how individual request response times influence system performance. An I/O request is considered *time-critical* if the generating thread blocks until the request is completed, e.g. a process is halted until a synchronous read request has been completed. *Time-limited* requests must be completed within a given amount of time otherwise they will become time-critical, for example file system read-aheads. Requests that do not require waiting times of the submitting process are classified as *time-noncritical*. Such disk I/O requests must be completed to maintain stable copies of nonvolatile storage, for example background flushes of asynchronous writes. Time-

noncritical requests can indirectly impact performance when interfering with the completion of more critical requests, thus model definition is complicated by the interaction between these different workload behaviors.

Extensive experimentation on emulated application workloads generated with the Filebench disk benchmark reveals that our methodologies can forecast successfully consolidation effects on I/O performance. Summarizing, the main contributions of this work in predicting consolidated workload performance are threefold.

1. Our methodology allows us to parameterize models based on data obtained inside VMs in isolation experiments. It requires very little information from the VMM thereby effectively treating the VMM as a black box. It also obviates the need to collect model training data for different VM consolidation scenarios.

2. The measurement-based decomposition model defines a set of simple analytical estimators to approximate mean performance of read/write disk requests separately.

3. The simulation model builds on the decomposition approach to derive fine-grained request response time distributions.

The remainder of the chapter is organized as follows. Section 4.2 lists the addressed research questions. Section 4.3 introduces the reference system for our study and Section 4.4 motivates the workload decomposition. Our models for heterogeneous workload consolidations are illustrated in Section 4.5, while Section 4.6 presents the corresponding validation results. Section 4.7 offers summary and conclusions.

4.2 Research Questions

This chapter addresses the following research questions:

- Can a block device I/O workload be decomposed for parameterization of separate models for read and write requests?

- How can block device I/O traces obtained within VMs in isolation benchmark experiments be used for performance prediction of heterogeneous workload consolidation scenarios?

- Is it possible to predict per request response times of heterogeneous workload consolidations for derivation of response time distributions?

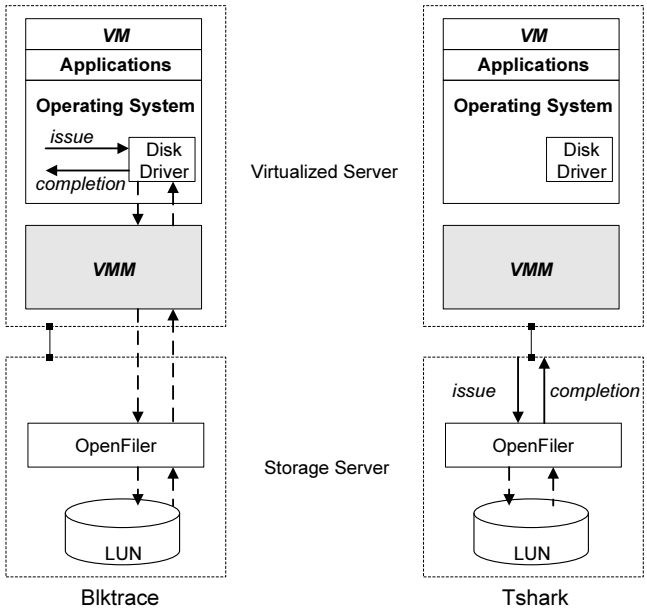


Figure 4.1: Monitoring of Reference System With the Blktrace and Tshark Tools.

4.3 Reference System

The reference system for this study is the same as that described in Section 3.5.1. In order to refine our model and analyze how the VMM submits I/O requests to the storage server, we extend the monitoring tool chain and use the network protocol analyzer *tshark* [14] to intercept iSCSI network packets. This extension allows us to obtain data for analysis of request arrival and completion times at the storage device. The tool is installed at the storage server and only collects the headers of iSCSI packets, which enables us to extract operational codes and control information without handling iSCSI payloads. We prevent monitoring data from impairing measurements by buffering into system RAM and only persist data to disk once benchmarking experiments are finished. Figure 4.1 illustrates how the monitoring extension complements the *blktrace* measurements described in Section 3.5.2 while still maintaining a black box view of the VMM layer.

The iSCSI protocol defines the direction of data transfer from the perspective of the initiator. Therefore write commands from the initiator, i.e. VMM, to the target, i.e. LUN, are considered outbound, while read commands are considered inbound [134]. We extract the disk I/O trace from the captured network packets by filtering for iSCSI operational codes defining read and write commands. The basic procedure for a read operation involves three steps:

1. The initiator sends a SCSI read command to the target, specifying the starting LBA and the number of contiguous blocks to transfer.

TS	OC	TYPE	IP_SRC	P_SRC	IP_DST	P_DST	BS	SIZE	LBA	TAG
...										
1.258608000	1	W	10.55.168.33	49168	10.55.168.22	3260	8	4096	316217895	75694649
1.258647000	49	N	10.55.168.22	3260	10.55.168.33	49168	0	0	0	75694649
1.258807000	5	N	10.55.168.33	49168	10.55.168.22	3260	0	0	0	75694649
1.258967000	37	N	10.55.168.22	3260	10.55.168.33	49168	0	0	0	75694644
1.259357000	1	R	10.55.168.33	49168	10.55.168.22	3260	16	8192	313038479	75694650
1.259457000	1	R	10.55.168.33	49168	10.55.168.22	3260	16	8192	314753783	75694651
1.259467000	1	R	10.55.168.33	49168	10.55.168.22	3260	16	8192	313039343	75694652
1.260804000	1	R	10.55.168.33	49168	10.55.168.22	3260	16	8192	313004399	75694653
1.263053000	33	N	10.55.168.22	3260	10.55.168.33	49168	0	0	0	75694649
...										

Figure 4.2: Example Disk I/O Trace From Tshark

2. The target sends the requested data to the initiator.

3. The target sends a SCSI status indicator to the initiator.

The procedure for a write operation involves four steps:

1. The initiator sends a SCSI write command to the target, specifying the starting LBA and the number of contiguous blocks that will be transferred.

2. The target sends an indication that it is ready to receive the data.

3. The initiator sends the data.

4. The target sends a SCSI status indicator to the initiator.

We define response times as the time difference between the first and the last step pertaining to a read/write operation. Figure 4.2 shows an excerpt from a captured tshark trace consisting of a nanosecond scale time stamp (TS), iSCSI operational code (OC), read/write request type (TYPE), IP-address of the request source (IP_SRC), port number of the request source (P_SRC), IP-address of the request destination (IP_DST), port number of the request destination (P_DST), number of blocks (BS), size of the request in byte (SIZE), LBA, and the iSCSI field *Initiator Task Tag* (TAG). The tag in the final field allows us to correlate multiple steps belonging to a single read/write operation.

Extracting iSCSI traces from network packets facilitates the computation of per request arrival and response times at the storage server. We synchronize system times between VMs and the storage server with the network time protocol (NTP). However, a 1:1 mapping of request data obtained with blktrace and tshark is not straightforward, since LBA address spaces differ at VM and VMM level.

4.4 Decomposition Analysis

We use the tshark tool to study how the reference system submits consolidated workloads to the storage device. Section 4.4.1 uses the monitoring data from the storage server for a static analysis of the arrival instant queue lengths seen by read and write requests. In Section 4.4.2 we show examples of how the arrival queue lengths dynamically change over time. Our investigation considers emulated mail, file, and web server application workloads, as described in Section 4.6.1.

4.4.1 Static Analysis of Arrival Queue Lengths

The I/O workload of the storage device consists of a mixture of read and write requests. Due to the mixed nature of consolidated workloads, ideally the system must minimize situations where time-noncritical write requests may interfere with time-critical reads. We use the *tshark* tool and the workloads described in Section 4.6.1 to observe how the arrival queue length at the storage server is partitioned between read and write requests, in order to study how our reference system submits requests of different types.

Figures 4.3 (a)-(f) show measurements from scenarios where only a single VM is running in isolation and convey the general impression that I/O requests of different read/write types are rarely submitted in the same time interval. Read requests of the file server workload mostly see small arrival queues of roughly five reads. Figure 4.3 (a) reveals a few cases where up to five writes are detected in the queue. The situation is similar for read requests from the mail server, as shown in Figure 4.3 (c). Arriving read requests of the web server workload, depicted in Figure 4.3 (e), may see some instances with larger queues containing up to 32 reads. However, we do not observe any cases where the web server read arrival queue contains write requests.

The write requests appear to be submitted in batch sizes of up to four or 16 as shown in Figures 4.3 (d) and (f) for mail and web server workloads, respectively. This property is not as evident for the file server workload, where Figure 4.3 (d) reveals mostly small write queues of up to five requests. It is important to note that in low utilized isolation scenarios queues seen by arriving write requests rarely contain any reads.

For our model of heterogeneous workload mixes we are especially interested to learn how the VMM submits consolidated I/O workloads to the disk. We present results of two VM consolidation scenarios in Figures 4.4 (a)-(f). Specifically, we consolidate one web and one mail server, denoted Web+Mail, one file and one mail server, denoted File+Mail, and one web and one file server, denoted Web+File.

Figure 4.4 (a) shows that a majority of arriving read requests in Web+Mail find only reads ahead of them. Mostly the utilization of the storage server is low, since only a small fraction

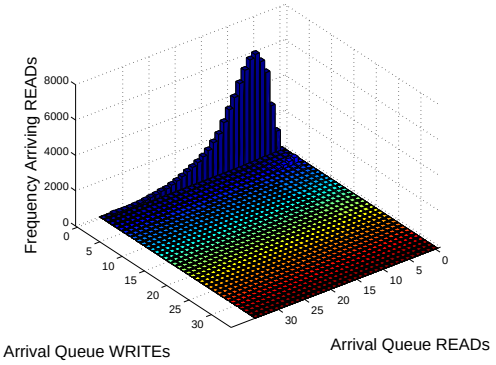
of the 32 available connections from the VMM to the storage server are being used. However, there is a significant number of instances where the maximum number of available connections are in use indicating high utilization periods. Interestingly, we also see a noticeable number of cases where the arrival queue contained 16 writes. In File+Mail, presented in Figure 4.4 (c), arriving reads do not see such large numbers of write requests in the arrival queue and additionally the number of reads in the queue is generally small. Figure 4.4 (e) suggests that high utilization periods of read requests might be a property of the web server workload, as we also observe situations where all 32 available connections from the VMM to the storage server are busy in Web+File.

Figure 4.4 (b) shows more variability in the arrival queue lengths for write requests. On most occasions arriving write requests in Web+Mail find a small number of reads, as well as other write requests roughly ranging in $[1; 16]$. These observations further suggest that a maximum of 16 write requests are batched from the VMM to the storage device. However, there also are noticeable frequencies where arriving write requests see large queues of read requests. These cases could indicate situations where batches of write and read requests are submitted within a short time interval. Such clashes of large amounts of read and write requests are not present in File+Mail. However, Figure 4.4 (d) illustrates high variability in the arrival queue where arriving writes do not only find other writes, but also up to 15 reads ahead of them. In Web+File the number of writes in the arrival queue is lower than in the previous cases, but Figure 4.4 (f) confirms that the VMM schedules write requests to the storage server whilst there are still read requests pending to be completed.

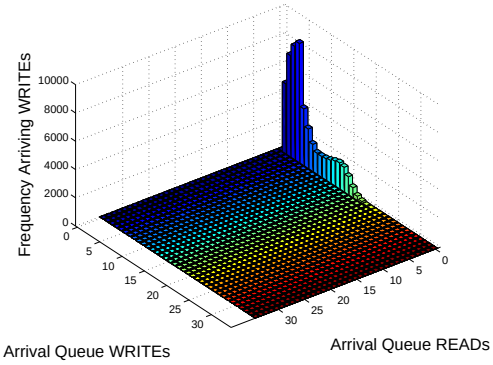
4.4.2 Time-varying Arrival Queue Lengths

To better understand the dynamic behavior of arrival queues, we randomly choose a 1s interval and illustrate how the queue at the storage server changes over time. Figures 4.5 (a) and (b) use the example of Web+Mail to show the time-varying arrival queue for read and write requests, respectively. The arrival patterns of both request types appear to be extremely bursty indicated by large numbers of requests submitted within a short time interval. Read requests are submitted in large bursts varying roughly in ranges of $[5; 32]$ that may utilize all the available 32 connections from the VMM to the storage device. Write requests appear to be scheduled in smaller batch sizes ≤ 16.

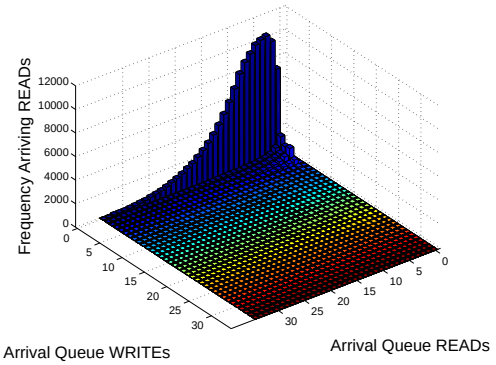
In the case of File+Mail we also observe the batched submission of requests. However, Figure 4.6 (a) shows that read request batch sizes are not as large, an observation that is confirmed by the analysis of the static arrival queue in Figure 4.4 (c). We can see some occasions where a burst of write requests finds a few scattered read requests and vice versa. Figure 4.6 (b) illustrates that arriving write requests find up to 16 read requests in the queue ahead of them.



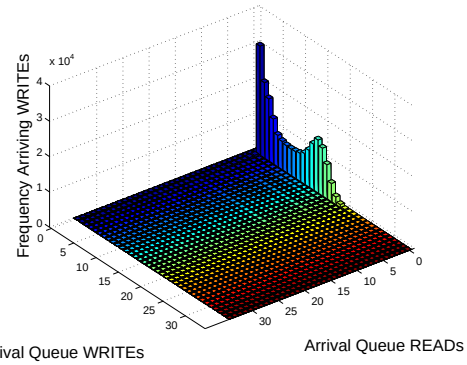
(a) Arrival Queue as Seen by Arriving Reads in File (64817 Arrivals)



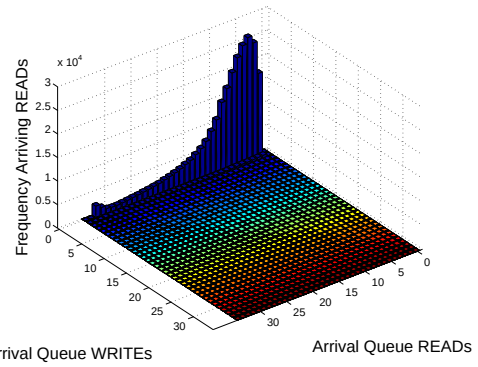
(b) Arrival Queue as Seen by Arriving Writes in File (50984 Arrivals)



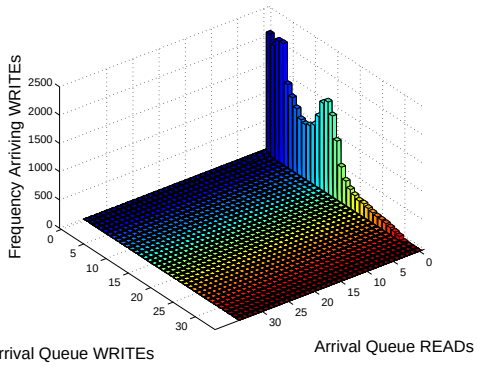
(c) Arrival Queue as Seen by Arriving Reads in Mail (131981 Arrivals)



(d) Arrival Queue as Seen by Arriving Writes in Mail (209408 Arrivals)

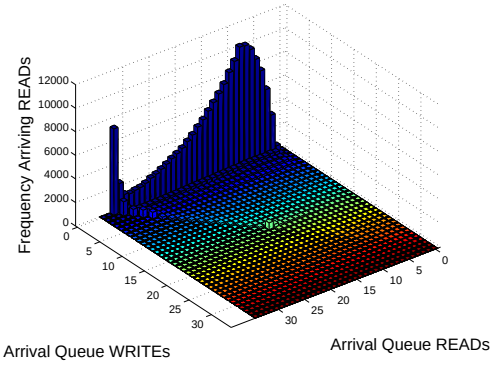


(e) Arrival Queue as Seen by Arriving Reads in Web (248966 Arrivals)

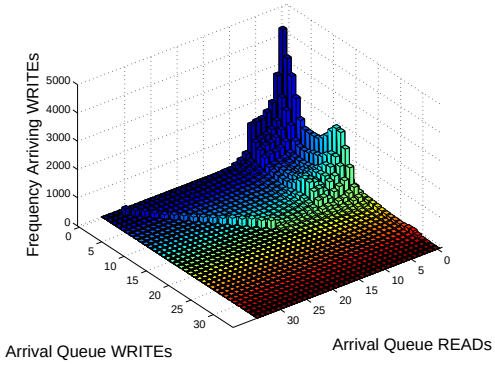


(f) Arrival Queue as Seen by Arriving Writes in Web (26739 Arrivals)

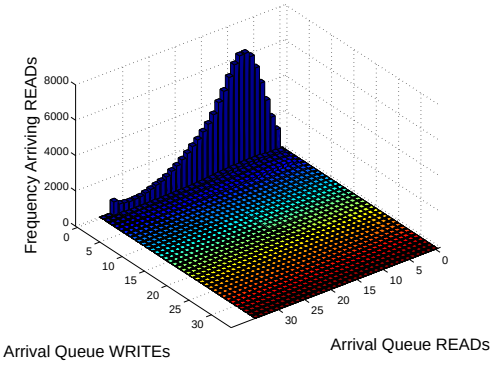
Figure 4.3: Arrival Queue Lengths From Single VM Experiments.



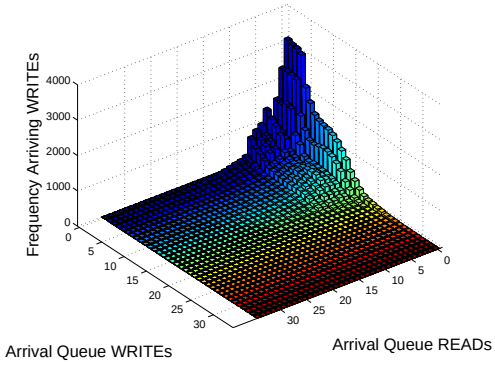
(a) Arrival Queue as Seen by Arriving Reads in Web+Mail (191786 Arrivals)



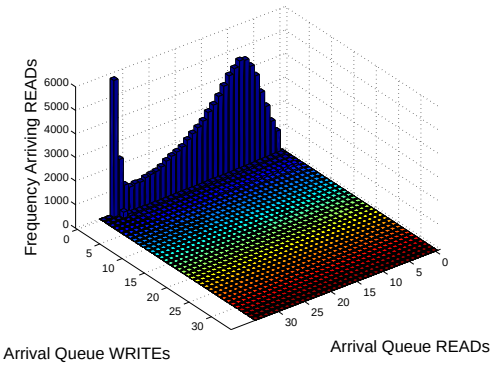
(b) Arrival Queue as Seen by Arriving Writes in Web+Mail (152899 Arrivals)



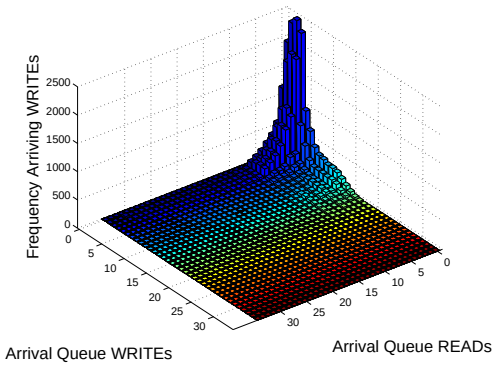
(c) Arrival Queue as Seen by Arriving Reads in File+Mail (103834 Arrivals)



(d) Arrival Queue as Seen by Arriving Writes in File+Mail (112663 Arrivals)



(e) Arrival Queue as Seen by Arriving Reads in Web+File (88317 Arrivals)



(f) Arrival Queue as Seen by Arriving Writes in Web+File (39639 Arrivals)

Figure 4.4: Arrival Queue Lengths From Two VM Consolidation Experiments.

However, the system seems to batch read and write requests independently: largely it either submits a burst of reads, or a burst of writes.

Summary. The static analysis of arrival queues has shown that in isolation measured queues at the storage device rarely contain requests of both read and write types. In consolidation this observation only holds for reads. This suggests that the VMM assigns a form of priority shares for read requests, e.g. similar to original UNIX systems (System 7) that used a disk request scheduler giving read requests non-preemptive priority. Thus read requests mostly compete with other read requests for the storage device, while the interference effects from writes is small. For write requests this property is not as evident, as our study finds situations where arriving write requests also find quantities of reads ahead of them at the storage device. The investigation of time-varying arrival queues confirms that the VMM submits read and write requests to the storage device in bursts.

4.5 Decomposition Model

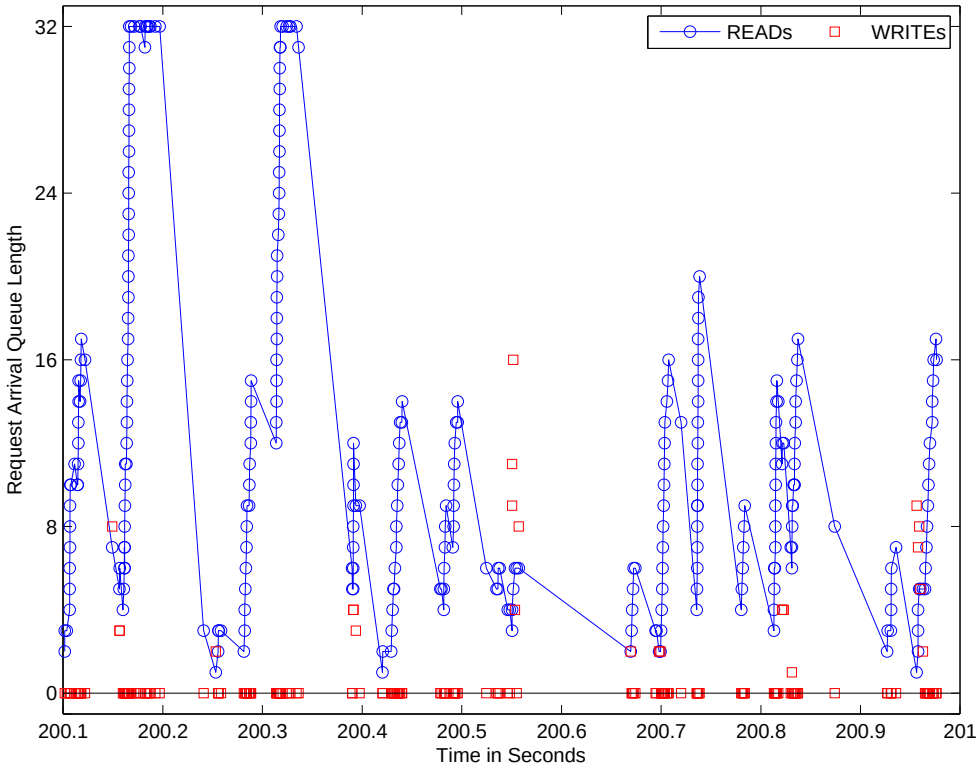
In this section we present models that decompose the workload into read and write requests to approximate performance metrics of heterogeneous disk I/O workload consolidations. Section 4.5.1 introduces a set of linear estimators to predict mean values of read/write mixes, throughputs, and response times. In Section 4.5.2 we present a simulation model that builds on the linear estimator concepts and outputs per request response times.

4.5.1 Linear Predictor Formulas

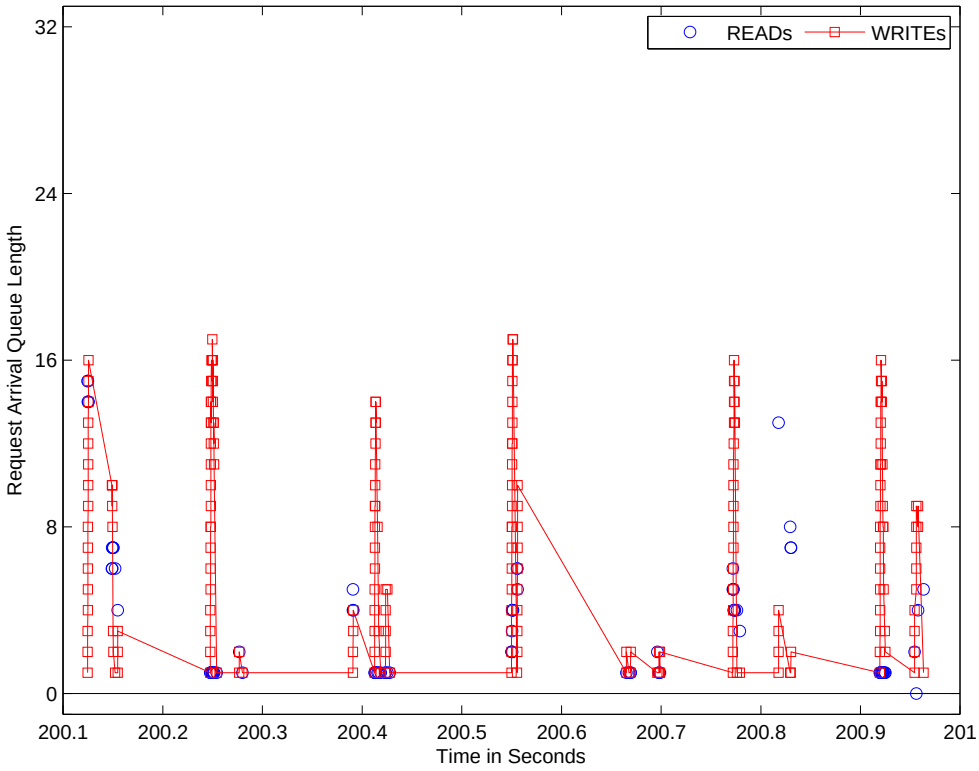
In this section, we discuss modeling the consolidation of heterogeneous workloads submitted from within VMs in ESX server using analytical predictors. We assume detailed information about each VM to be available from the isolation experiments, and we focus on the prediction of the expected I/O performance of the VMs in consolidation. Our performance prediction methodology considers the following performance metrics. We only include definitions for two VM consolidation scenarios as the generalization is obvious:

- T_i^R: mean throughput of VM i's read requests in isolation
- $T_{i,j}^R$: mean throughput of VM i's read requests in consolidation with VM j
- $X_{i,j}^R$: throughput of read requests, originating from any VM, in the consolidation of VMs i and j

A similar notation T_i^W, $T_{i,j}^W$, and $X_{i,j}^W$ is used for write requests. We also introduce the following derived quantities:

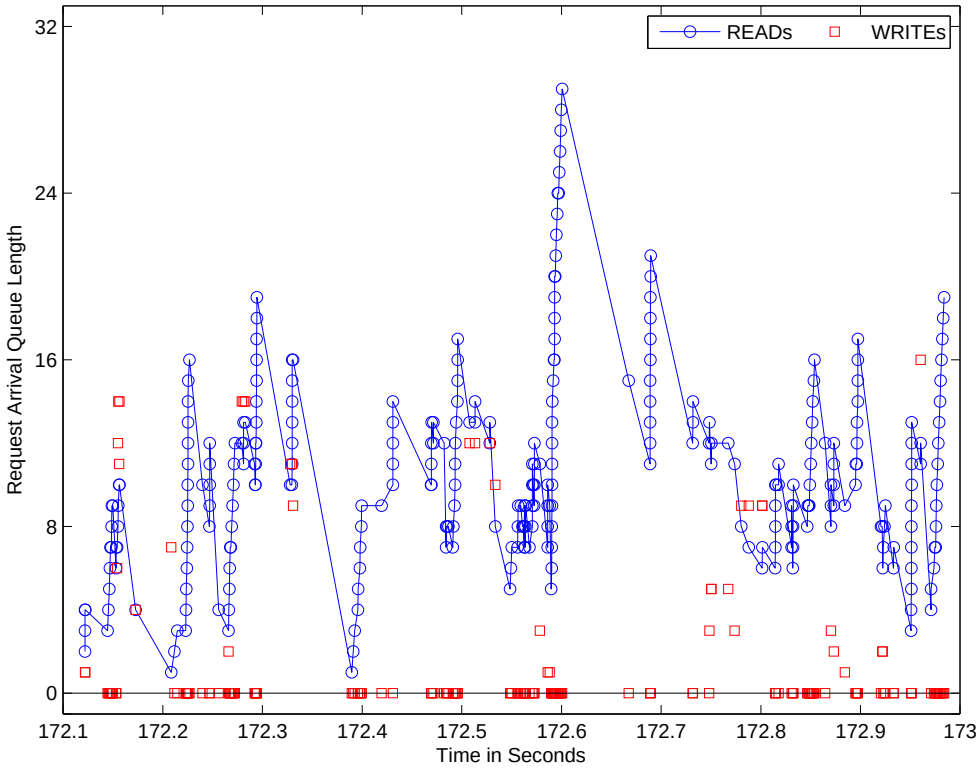


(a) Arrival Queue as Seen by Arriving Reads

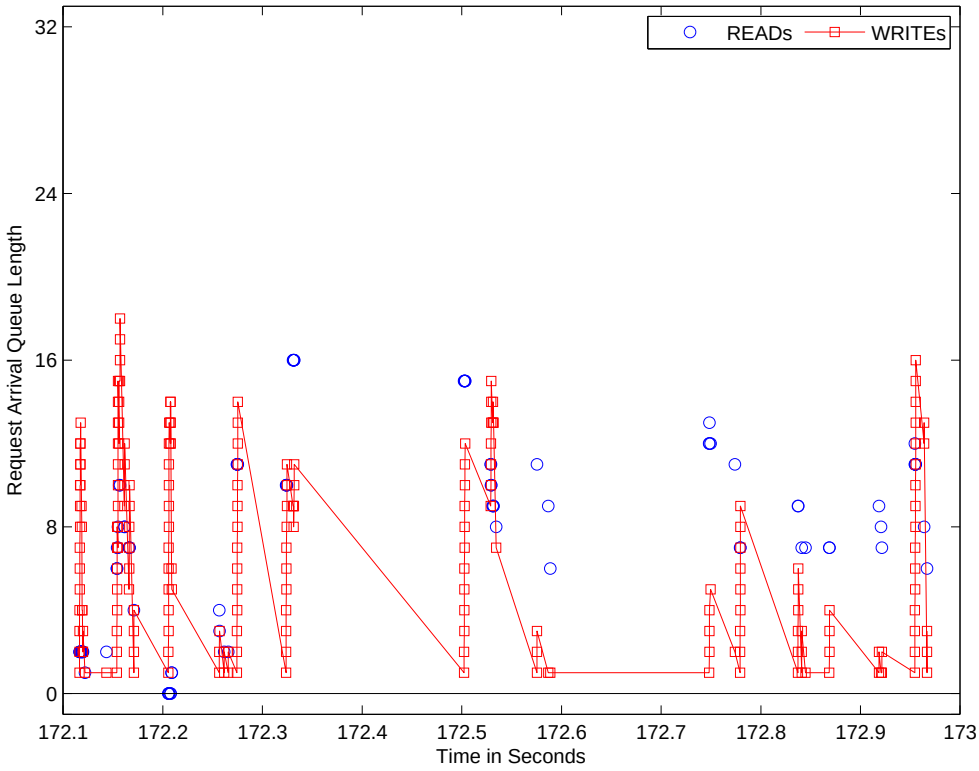


(b) Arrival Queue as Seen by Arriving Writes

Figure 4.5: Time-Varying Analysis of Arrival Queue Lengths in Web+Mail.



(a) Arrival Queue as Seen by Arriving Reads



(b) Arrival Queue as Seen by Arriving Writes

Figure 4.6: Time-Varying Analysis of Arrival Queue Lengths in File+Mail.

- $T_i = T_i^R + T_i^W$: mean throughput of VM i in isolation
- $T_{i,j} = T_{i,j}^R + T_{i,j}^W$: mean throughput of VM i in consolidation with VM j
- $\alpha_i^R = T_i^R / T_i$: relative throughput fraction of read requests for VM i in isolation
- $\alpha_{i,j}^R = T_{i,j}^R / T_{i,j}$: relative throughput fraction of VM i's read requests in consolidation with VM j
- $\beta_{i,j}^R = (T_{i,j}^R + T_{j,i}^R)/(T_{i,j} + T_{j,i})$: mix of read requests, originating from any VM, in the consolidation of VMs i and j

Similar quantities α_i^W, $\alpha_{i,j}^W$, and $\beta_{i,j}^W$ are defined for write requests. In addition, for response times we define indexes C_i^R, $C_{i,j}^R$ with similar meaning of the corresponding indexes for throughputs. We now propose three classes of linear estimators for request read/write mixes, throughputs, and response times in consolidation.

4.5.1.1 Approximation of Consolidation Read/Write Mixes

Let us first introduce the following linear predictor for the consolidation read/write mixes $\beta_{i,j}^R$ and $\beta_{i,j}^W$.

$$\beta_{i,j}^R \approx \alpha_i^R \left(\frac{T_i}{T_i + T_j} \right) + \alpha_j^R \left(\frac{T_j}{T_i + T_j} \right) \tag{4.1}$$

$$\beta_{i,j}^W \approx \alpha_i^W \left(\frac{T_i}{T_i + T_j} \right) + \alpha_j^W \left(\frac{T_j}{T_i + T_j} \right) \tag{4.2}$$

Justification. Let L be the length of the period over which the system was observed and denote with n_i^R (resp. n_i^W) the number of read (resp. write) requests completed by the system in L. Then by definition $T_i^R = n_i^R/L$, $T_i^W = n_i^W/L$, and thus $\alpha_i^R = n_i^R/(n_i^R + n_i^W)$, $\alpha_i^W = n_i^W/(n_i^R + n_i^W)$. Using a similar notation for consolidation we get $\alpha_{ij}^R = n_{i,j}^R/(n_{i,j}^R + n_{i,j}^W)$, $\alpha_{ij}^W = n_{i,j}^W/(n_{i,j}^R + n_{i,j}^W)$. Then by definition

$$\begin{aligned}
\beta_{i,j}^R = \frac{T_{i,j}^R + T_{j,i}^R}{T_{i,j} + T_{j,i}} &= \frac{n_{i,j}^R + n_{j,i}^R}{n_{i,j}^R + n_{i,j}^W + n_{j,i}^R + n_{j,i}^W} \\
&= \left(\frac{n_{i,j}^R + n_{i,j}^W}{n_{i,j}^R + n_{i,j}^W + n_{j,i}^R + n_{j,i}^W} \right) \left(\frac{n_{i,j}^R}{n_{i,j}^R + n_{i,j}^W} \right) \\
&\quad + \left(\frac{n_{j,i}^R + n_{j,i}^W}{n_{i,j}^R + n_{i,j}^W + n_{j,i}^R + n_{j,i}^W} \right) \left(\frac{n_{j,i}^R}{n_{j,i}^R + n_{j,i}^W} \right)
\end{aligned}$$

where we can identify the terms $\alpha_{i,j}^R = n_{i,j}^R/(n_{i,j}^R + n_{i,j}^W)$ and $\alpha_{j,i}^R = n_{j,i}^R/(n_{j,i}^R + n_{j,i}^W)$. Consider now the approximation $\alpha_i^R \approx \alpha_{i,j}^R$, which assumes that the relative throughput fraction of

reads incoming from VM i is the same in isolation and consolidation. This approximation is accurate if the arrival process of VM i is loosely dependent on the overheads of consolidation. Using this approximation we get

$$\beta_{i,j}^R = \left(\frac{n_{i,j}}{n_{i,j}+n_{j,i}}\right)\alpha_i^R + \left(\frac{n_{j,i}}{n_{i,j}+n_{j,i}}\right)\alpha_j^R \\ = \left(\frac{T_{i,j}}{T_{i,j}+T_{j,i}}\right)\alpha_i^R + \left(\frac{T_{j,i}}{T_{i,j}+T_{j,i}}\right)\alpha_j^R$$

where the last passage follows by first scaling numerator and denominators by L. The final formula is obtained by further approximating the throughput ratios in consolidation by the ratios of T_i and T_j in isolation. This corresponds to the assumption that a common overhead factor c_{oh} exists for the two VMs such that

$$\left(\frac{T_{i,j}}{T_{i,j}+T_{j,i}}\right) = \left(\frac{c_{oh}T_i}{c_{oh}T_i + c_{oh}T_j}\right) = \left(\frac{T_i}{T_i+T_j}\right)$$

The justification for $\beta_{i,j}^W$ follows in a similar way.

4.5.1.2 Approximation of Consolidation Throughputs

Using a similar justification as the one for read/write mixes, we define the following heuristic for estimating the total throughputs of read and write requests in consolidation

$$X_{i,j}^R \approx T_i^R\left(\frac{T_i}{T_i+T_j}\right) + T_j^R\left(\frac{T_j}{T_i+T_j}\right) \\ X_{i,j}^W \approx T_i^W\left(\frac{T_i}{T_i+T_j}\right) + T_j^W\left(\frac{T_j}{T_i+T_j}\right) \tag{4.3}$$

4.5.1.3 Approximation of Consolidation Response Times

The approximation we propose involves using the response times and arrival queue-lengths collected with the *blktrace* or *tshark* tools in isolation experiments to estimate the expected response times in consolidation. Our measurements indicate that quantities from both tools are very similar in isolation. Let S_i^R and S_i^W be the estimated service times of read and write requests in isolation. Assuming first-come first-served as an approximation of the scheduling policy at the disk, we use the following expression:

$$S_i^R \approx C_i^R/(1+A_i^R), \tag{4.4}$$

where $1+A_i^R$ is the queue-length seen upon arrival by a read request at ESX including the arriving job itself [126]. Equivalently, the arrival queue-length can be considered at the storage

server if ESX data is not directly available, since we found that in isolation the difference between the two measurements is often negligible. Then we estimate the expected response time for a read request in consolidation as

$$C_{i,j}^R \approx C_i^R + S_j^R A_j^R \tag{4.5}$$

This approximation assumes that the overheads for reads in consolidation for VM i are only due to the interference with reads issued by the other VM j; write requests do not interfere with the response times of reads for VM i. Figures 4.4 (a), (c), and (e) support this assumption, as the large majority of read arrivals only find requests of the same type in the system.

For write requests the non-interference property is not as evident, see Figures 4.4 (b), (d), and (f) where some reads can be found in the system by an arriving write. However, it is interesting to observe that such reads rarely exceed 5-10 in number, except for a few rare events, e.g. depicted in the middle of Figure 4.4 (b) where tens of both reads and writes are found on arrival to the system. Our approximation ignores such rare events and we leave this extension of our estimator for future work. Based on this approximation approach, we define the mean response time of write requests in consolidation as

$$C_{i,j}^W \approx C_i^W + S_j^W A_j^W \tag{4.6}$$

Finally, similar expressions are defined for the response times of VM j, i.e., $C_{j,i}^R$ and $C_{j,i}^W$. We refer to Table 4.1 for a summary of parameters and estimators of the linear predictor formulas.

4.5.2 Simulation Model

We complement the decomposed modeling methodology with a simulation model that not only facilitates prediction of mean response times in consolidation, but additionally allows us to approximate the response time distribution. In some situations prediction of response time distributions might be preferable to mean values, e.g. in cases where certain percentiles specified in Service Level Agreements (SLAs) need to be satisfied.

Our model follows the concepts of the linear estimators introduced in Section 4.5.1 and focuses on the mostly synchronous and performance critical read requests [171]. To capture the feedback effects between read request completions and arrivals we resort to a multiclass closed queueing model as sketched in Figure 4.7. Feedback effects in disk workloads can have significant impact on performance [90]. Requests submitted from individual VMs are distinguished in K separate classes. Based on the assumptions and approximations in Section 4.5.1.3 the simulation model uses a single server with FCFS scheduling.

To maintain the burstiness in the arrival process throughout the simulation run the model

Table 4.1: Summary of Input and Output Parameters for the Decomposition Model Linear Estimators When Predicting Read Requests in Two VM Consolidation Scenarios. The Notation for Write Requests is Similar.

Input	T_i, T_j	Mean throughput of VM i (resp. VM j) in isolation.
	T_i^R, T_j^R	Mean throughput of VM i (resp. VM j) read requests in isolation.
	α_i^R, α_j^R	Relative throughput fraction of read requests for VM i (resp. VM j) in isolation.
	C_i^R, C_j^R	Mean response time of VM i (resp. VM j) read requests in isolation.
	A_i^R, A_j^R	Mean arrival queue lengths of VM i (resp. VM j) read requests in isolation.
Output	$\beta_{i,j}^R$	Mix of read requests from any VM in consolidation of VMs i, j.
	$X_{i,j}^R$	Mean throughput of read requests from any VM in consolidation of VMs i, j.
	$T_{i,j}^R, T_{j,i}^R$	Mean throughput of VM i (resp. VM j) read requests in consolidation with VM j (resp. VM i).
	$C_{i,j}^R, C_{j,i}^R$	Mean response time of VM i (resp. VM j) read requests in consolidation with VM j (resp. VM i).

population consists of *BatchCustomers*, denoted bc^R and bc^W, that are forked and joined in custom model elements. Before arriving at the queue each *BatchCustomer* enters a *BatchForkStation*, where it is forked into a batch, such that:

$$(n \in \mathbb{N}, 1 \leq n \leq b^R) : A(c_k^{R,i,n}) = A(bc_k^{R,i}), \tag{4.7}$$

where b^R is a read request batch size sampled from a probability distribution, $c_k^{R,i,n}$ is a forked read request of class k, $A(c_k^{R,i,n})$ is the arrival time of a forked request, and $A(bc_k^{R,i})$ is the arrival time of a *BatchCustomer*. After leaving the service station all elements of the previously forked *BatchCustomer* are re-joined in the *BatchJoinStation* model element. We compute per request response times for each $c_k^{R,i,n}$ as the time difference between request completion $E(c_k^{R,i,n})$ and arrival $A(c_k^{R,i,n})$. A similar notation b^W, $c_k^{W,i,n}$, $A(c_k^{W,i,n})$, $A(bc_k^{W,i})$, $E(c_k^{W,i,n})$ is used for write requests.

4.5.2.1 Model Parameterization

The parameterization process of the simulation model consists of two steps as outlined in Algorithm 3. First we use measurements obtained with *blktrace* in a single representative isolation benchmark experiment to determine disk I/O request batch sizes, batch think times, and service

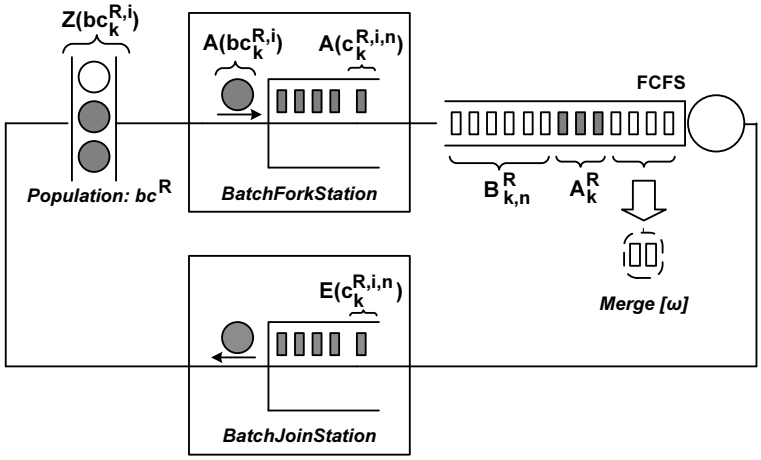


Figure 4.7: Customized Closed Queueing Model.

Algorithm 3 Parameterization of Simulation Model

#step 1: parameters from isolation measurements
$\Pr[|B_{k,n}^R| = b^R] \leftarrow measured\ distribution\ of\ batch\ sizes$
$Trace[Z_{k,n}^R] \leftarrow measured\ trace\ of\ batch\ think\ times$
$Exp[D(c_{k,a_k}^R)] \leftarrow exponential\ service\ as\ function\ of\ load$
#step 2: estimate population parameter
$X_k^{R,est\_lin} \leftarrow throughput\ estimate\ from\ linear\ predictor$
$X_k^{R,est\_sim} \leftarrow throughput\ estimate\ from\ simulation$
$N_k^{R,bc} = 1 \leftarrow init\ model\ population$
while $X_k^{R,est\_sim} < X_k^{R,est\_lin}$ **do**
  $simulate$
  $increase\ N_k^{R,bc}$
end while
$lower = N_k^{R,bc} - 1 \leftarrow lower\ bound\ population\ size$
$upper = N_k^{R,bc} \leftarrow upper\ bound\ population\ size$

times. Secondly, we find an approximation of the model population by modulating this parameter until throughputs in simulation match a throughput estimate computed with the linear predictor formulas. See the following paragraphs for detailed definitions of model parameters.

Batch Sizes. The batched submission of I/O requests is an important modeling parameter as bursty arrival patterns can have a large impact on system performance [97]. We have found the batch size parameter to largely influence the quality of the modeling result and initially determine this quantity from measured disk I/O request arrival traces. The elements of a batch are defined as:

$$\forall c_k^{R,i} \in B_{k,n}^R : \text{diff}\{A(c_k^{R,i+1}), A(c_k^{R,i})\} \leq t_{val}, \tag{4.8}$$

where batch $B^R_{k,n}$ holds all read arrivals $c^{R,i}_k$ that have interarrival times ranging within a time-out constraint t_{val}. We specify t_{val} using the custom technique described in Section 3.7.2.4 and report its value in Section 4.6.2.2. Similar quantities $c^{W,i}_k$, $B^W_{k,n}$, $A(c^{W,i}_k)$ are defined for write requests.

Once the batches in the arrival time series are found, we define the size of a batch $B^R_{k,n}$ as $|B^R_{k,n}|$ and compute the probability distribution $\Pr[|B^R_{k,n}| = b^R]$ for each workload. The distribution of batch sizes constitutes a parameter of the *BatchForkStation* model element: A batch size b^R is uniformly sampled for each arrival of a read request *BatchCustomer* before it is forked into b^R requests.

Think Times. The model captures think times as the time between completion of batch n and arrival of the following batch $n+1$. We compute this parameter based on the batch population as:

$$Z^R_{k,n} = \text{diff}\{\min(A(c^{R,i}_k), c^{R,i}_k \in B^R_{k,n+1}), \max(E(c^{R,i}_k), c^{R,i}_k \in B^R_{k,n})\}, \tag{4.9}$$

where the think time $Z^R_{k,n}$ is the difference between the largest completion time $E(c^{R,i}_k)$ of batch $B^R_{k,n}$ and the smallest arrival time of any request belonging to the successive batch $B^R_{k,n+1}$. We define similar quantities $Z^W_{k,n}$, $E(c^{W,i}_k)$ for write requests. In the simulation model we use measured think time traces to assign a think time $Z(bc^{R,i}_k)$ for each *BatchCustomer* $bc^{R,i}_k$.

Service Times. We approximate the service requirement of requests using the well-known mean value analysis (MVA) expression introduced in (4.4). As Section 4.6.2.1 reveals our workload characterization study has shown that service times are load dependent under the simplifying assumption of FCFS scheduling. We estimate the mean request service time as a function of the queue seen on arrival by

$$D(c^R_{k,a_k}) \approx \frac{1}{n}\sum_{i=1}^{n} C(c^{R,i}_k)/(1+A^R_k), a_k = (1+A^R_k), \tag{4.10}$$

where $D(c^R_{k,a_k})$ is the mean service time of read requests given the arrival instant queue length a_k and measured response time $C(c^{R,i}_k)$. For write requests we define the quantity $D(c^W_{k,a_k})$. During simulations of consolidation scenarios each read request arrival $c^{R,i,n}_k$ observes the arrival queue length of requests of its own class, A^R_k, and samples an exponentially distributed service time with mean $D(c^R_{k,a_k})$, where $a_k = 1 + A^R_k$.

Population. The model is parameterized with a population of *BatchCustomers*. Before entering the queueing station each *BatchCustomer* is forked into a number of individual requests and thus represents a batch arrival. Determining the population size from measurements, e.g.

Table 4.2: Measured Throughputs (cmd/s) in Isolation at VM and Storage Server Level.

Workload	*VM*		*Storage Server*			
	Blktrace		Tshark		Ratio	
	R	W	R	W	R	W
File	330	237	198	153	0.60	0.65
Mail	245	370	245	380	1.00	1.03
Web	470	44	462	53	0.98	1.20

the number of benchmark application threads, is difficult, since a batch of requests submitted by in-VM schedulers might consist of requests originating from multiple threads. In order to approximate the appropriate population size in consolidation based on isolation measurements only, we leverage results obtained from the previously introduced linear predictor formulas. Specifically, we predict consolidation throughputs as defined in Section 4.5.1.2 and then modulate the population size in simulations until simulation throughputs closely match the analytical prediction.

Since the configurable population size constitutes batch arrivals rather than individual requests, it might not always be possible to exactly match analytically estimated with simulated throughputs. In such cases we determine upper and lower bounds on population sizes and record response time predictions via linear interpolation between these bounds.

4.5.2.2 Model Calibration Methodology

During benchmark experiments we found that in-VM throughput measurements can significantly differ from throughputs measured at the storage server. In the absence of detailed knowledge of VMM internals we explain this behavior with optimization operations along the storage I/O path. Table 4.2 shows measurements averaged over 15 benchmark runs and reveals that file server throughputs are especially affected by these operations, where measured throughputs at the storage server are reduced to roughly 60% and 65% compared to quantities recorded inside VMs for read and write requests, respectively. Throughputs for web and mail server read workloads are monitored at almost identical quantities across the system stack. Interestingly, we record 20% larger throughputs for web server write requests at the storage server.

We model this behavior with a heuristic calibration technique similar to Section 3.6.3.1. However, instead of parameterizing the queueing station with a static merge value ω we estimate this parameter dynamically for each batch arrival. The ω value is based on the throughput

Table 4.3: Summary of Input and Output Parameters for the Decomposition Simulation Model When Predicting Read Requests. The Notation for Write Requests is Similar.

Input	$\Pr[\lvert B^R_{k,n}\rvert = b^R]$	Measured distribution of request batch sizes from VM in isolation.
	$Z(bc^{R,i}_k)$	Trace of request batch think times measured from VM in isolation.
	$D(c^R_{k,a_k})$	Mean measured request service time for arrival queue a_k.
	X^{R,est_lin}_k	Throughput estimate in consolidation from linear predictor formulas.
	ω	Merge value parameter from measured throughput ratio.
	$N^{R,bc}_k$	Population of *BatchCustomers* derived through calibration.
Output	$C(c^{R,i}_k)$	Predicted request response times in consolidation.
	X^{R,est_sim}_k	Predicted throughput in consolidation.

ratios measured in isolation, as well as the size b^R of an arriving batch:

$$\omega = b^R \left(1 - \frac{T^R_{k,san}}{T^R_{k,vm}}\right), \quad T^R_{k,vm} \geq T^R_{k,san}, \tag{4.11}$$

where $T^R_{k,vm}$ and $T^R_{k,san}$ are the throughputs measured at VM and storage server level, respectively. Thus we merge an equivalent percentage of requests from each batch arrival as we observe throughput reductions in our isolation measurements. Table 4.3 offers a summary of input and output parameters for the decomposition simulation model.

4.6 Validation Experiments: Decomposition Model

In this section we validate the accuracy of the proposed decomposition model based on a number of benchmark experiments. We first introduce and characterize the workload types considered before showing measurement results and prediction errors.

4.6.1 Workload Generation

We conduct our study using emulated disk workloads of mail, web, and file server type applications. Workloads are generated with FileBench [26], a framework for measuring and comparing file system performance. We maintain the default workload specification and use the recommended parameters for small configurations (50000 files in the initial file set). The presented measurements are gathered during a series of benchmarking experiments, each consisting of

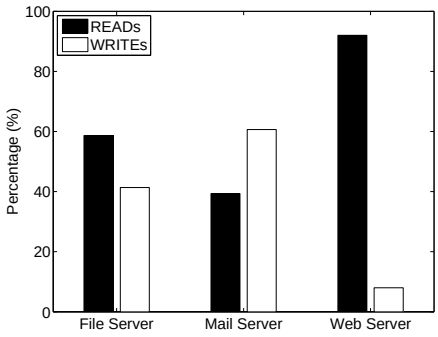


Figure 4.8: Percentages of Read/Write Mixes for File, Mail, and Web Server Workloads.

10 runs of 300s length. See Appendix D for the utilized benchmark configurations. We report results as the means over all 10 iterations or based on a representative run. In total we conduct 420 benchmark runs and collect roughly 53 million measurement samples. Appendix C provides a more detailed summary of benchmark statistics. Experiments are conducted with a single VM running in isolation and with two or three VMs on the same server in consolidation. For example, we consolidate one web and one mail server, denoted Web+Mail, and one web and two file servers, denoted Web+File+File.

Figure 4.8 shows the measured read/write mixes for the considered workloads in isolation. File and web server workloads contain a majority of reads, with the latter only comprising a small write portion of less than 10%. On the contrary, the mail server workload mix has a larger fraction of writes. Summarizing, the considered application workloads feature distinct read/write mixes to allow a wide range of evaluation scenarios for our decomposed modeling approach.

4.6.2 Workload Characterization

In this section we show a characterization of measured disk I/O request service times, batch sizes, and batch think times that are used for model parameterization.

4.6.2.1 Service Times

We estimate the mean service time of disk I/O requests based on measurements of response times and arrival queue lengths obtained in isolation as described in Section 4.5.2.1. Our observations show that service times during phases of lower loads are larger than during phases of higher loads. We explain this behavior due to the disk device's capabilities to serve large batches of requests faster than small amounts of individual requests, i.e. the per request service times of a batched sequential read versus service requirements of multiple random reads.

Table 4.4: Mean in ms, Standard Deviation (std), and Coefficient of Variation (cv) Statistics for Read Request Service Times, Batch Sizes, and Batch Think Times as Measured in Isolation with Blktrace. t_{val} in ms Specifies the Measured Batch Timeout Constraint From Isolation.

Workload	*Service Time*			*Batch Size*			*Batch Think Time*			*Batch T-Constraint*
	mean	std	cv	mean	std	cv	mean	std	cv	t_{val}
File	2.94	4.39	1.50	5.07	4.85	0.96	151	188	1.24	2.35
Mail	2.66	2.82	1.06	5.55	5.39	0.97	120	104	0.86	1.03
Web	1.58	2.43	1.56	5.79	6.97	1.20	141	88.4	0.63	2.34

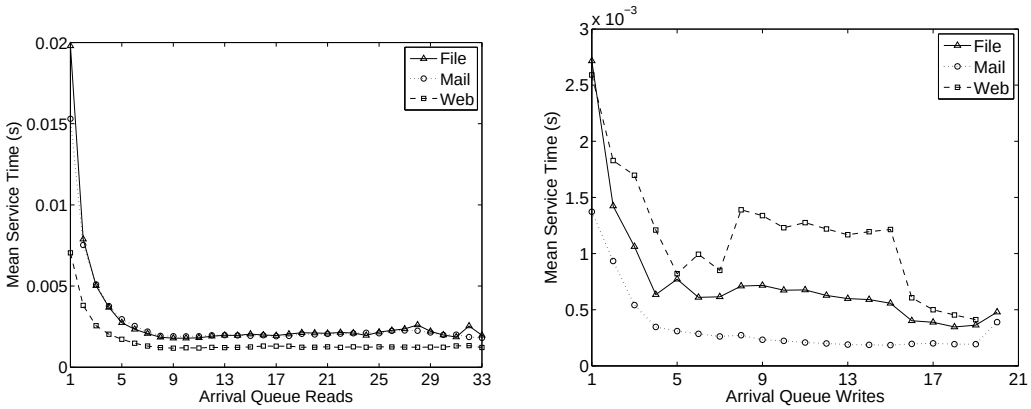
Table 4.5: Mean in ms, Standard Deviation (std), and Coefficient of Variation (cv) Statistics for Write Request Service Times, Batch Sizes, and Batch Think Times as Measured in Isolation with Blktrace. t_{val} in ms Specifies the Measured Batch Timeout Constraint From Isolation.

Workload	*Service Time*			*Batch Size*			*Batch Think Time*			*Batch T-Constraint*
	mean	std	cv	mean	std	cv	mean	std	cv	t_{val}
File	0.75	4.50	6.01	4.00	2.96	0.72	147	283	1.9	0.12
Mail	0.35	0.93	2.70	4.84	6.77	1.40	175	118	0.67	0.70
Web	1.16	8.01	6.93	4.62	4.36	0.94	884	1574	1.78	0.16

This may be a mechanical property of the disk serving multiple requests with a single motion of the disk arm. Tables 4.4 and 4.5 show service time statistics for read and write requests, respectively.

Figure 4.9 (a) reveals that read request service requirements of file and mail server workloads are almost identical. Service times decrease with increasing load and stabilize roughly at arrival queue lengths ≥ 9. Web server disk requests generally take less time to serve. However, the distribution follows the same trend as workloads of types file and mail. Service demands of read requests in the queueing model are assumed to be exponentially distributed. We have found this to be a good approximation since our measurements roughly match the theoretical exponential of $CV = 1$: The CV of measured service times ranges in $CV^R_{a_k} = [0.39; 3.13]$ for all $D(c^R_{k,a_k})$.

The service times of write requests are also load dependent, but we find higher variability in the measured quantities. Figure 4.9 (b) shows that service requirements of writes are considerably smaller than those of read requests. Write requests of the mail server load display the clearest trend and stabilize with increasing load roughly around arrival queue lengths of ≥ 5. File server requests take longer to serve and the distribution is not as stable as in the case of mail. We record the largest write request service times for the web server. The unclear trend of the web server service time distribution might be due to interference effects from read requests and the small amount of write samples in the measurement: The portion of write requests in the web server workload is $\leq 10\%$. We model the high variability of write request service demands with a hyper-exponential distribution. The CV of measured service times ranges mostly in $CV^W_{a_k} = [3.54; 10.8]$, except for a few cases with CVs close to 1 for mail server writes.



(a) Read Requests (b) Write Requests

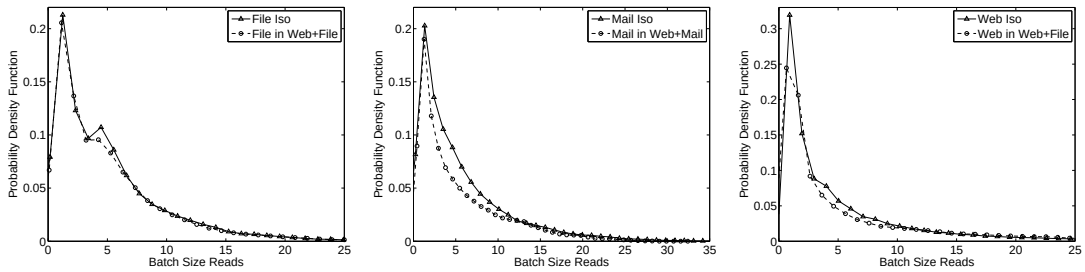
Figure 4.9: Load Dependent Service Times for File, Mail, and Web Server Workloads.

4.6.2.2 Batch Sizes

The batched submission of disk requests may result in bursty arrival patterns that have large impacts on system performance. Our model tries to capture such bursts by first detecting batches in isolation measurements using the technique described in Section 3.7.2.4 and secondly using the batch size distribution as a model parameter. Tables 4.4 and 4.5 show statistics of our batch detection technique and batch sizes for read and write requests, respectively. Mean batch sizes of read requests are marginally larger than the ones of write requests. Both request types also share a low variability of measured batch sizes with CVs close to 1.

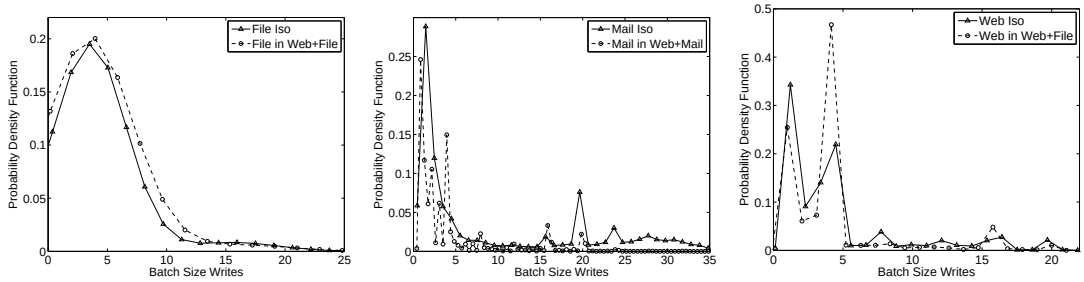
We are especially interested to learn how batch sizes are impacted by workload consolidation, since we use isolation batch size measurements to parameterize simulations of consolidated workloads. The examples in Figures 4.10 (a)-(c) show that batch size distributions of read requests are similar in isolation and consolidation. Distributions of file and web server requests almost converge, while the probabilities for larger read request batch sizes of the mail server are slightly higher in isolation compared to Web+Mail.

Consolidation also appears to have only minor effects on the batch sizes of write requests. Figure 4.11 (a) illustrates that the distributions of file server batch sizes are similar in isolation and Web+File. We observe more variability in the cases of mail and web server workloads. However, Figures 4.11 (b) and (c) illustrate that batch size distributions follow the same trend when comparing isolation measurements to the consolidation examples Web+Mail and Web+File, respectively.



(a) File (b) Mail (c) Web

Figure 4.10: Measured Distribution of Read Request Batch Sizes for File, Mail, and Web Server Workloads.



(a) File (b) Mail (c) Web

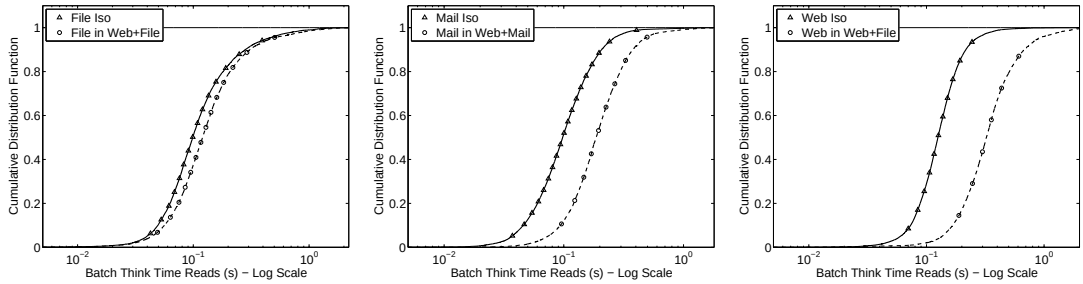
Figure 4.11: Measured Distribution of Write Request Batch Sizes for File, Mail, and Web Server Workloads.

4.6.2.3 Batch Think Times

We measure think times of batches as the time between the largest request completion time of batch n and the smallest request arrival time of batch $n + 1$. Interestingly, measurements indicate that mean think times are similar for both request types across workloads, except for the web server workload with the significantly increased value of 884ms. Variabilities in think times are quite low with CV ranging in $[0.63; 1.9]$. See Tables 4.4 and 4.5 for think time details of read and write requests, respectively.

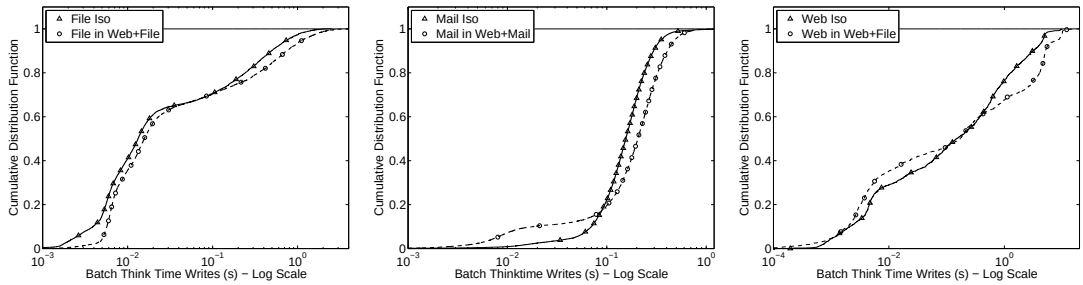
Consolidation seems to affect read request batch think times of the considered workloads to different extents. Figure 4.12 (a) conveys a good match of batch think time distributions in isolation and Web+File consolidation for the file server workload. Conversely, in Figures 4.12 (b) and (c) we find higher probabilities for larger consolidation think times in the cases of mail and web server reads, respectively. This discrepancy may be explained with feedback effects of the mostly synchronous read requests, e.g. when the submission of a read batch relies on the completion of another batch of reads.

Figures 4.13 (a)-(c) show that isolation and consolidation batch think time distributions of write requests match considerably better than the previously discussed read request batch think



(a) File (b) Mail (c) Web

Figure 4.12: Measured Distribution of Read Batch Think Times for File, Mail, and Web Server Workloads.



(a) File (b) Mail (c) Web

Figure 4.13: Measured Distribution of Write Batch Think Times for File, Mail, and Web Server Workloads.

time distributions. Our study shows slightly larger probabilities for increased write batch think times in the presented consolidation examples. We take these findings as further indication that the majority of write requests in the considered workloads are asynchronous and therefore not subjected to feedback effects.

Summary. Our workload characterization study has shown that service times of read and write requests are load dependent under the simplifying assumption of FCFS scheduling at the storage device. Furthermore, the distributions of batch sizes are similar in isolation and consolidation thus consolidation only has minor effects on the sizes of bursts. As our model samples batch sizes from a distribution measured in isolation, this modeling abstraction seems to be a good approximation of system behavior. Think times of write batches are only marginally impacted by workload consolidation. Using batch think time isolation measurements for model parameterization therefore is a valid approximation. For read request batch think times this observation is not as clear as some workloads are more affected than others.

Table 4.6: Response Time (ms), Throughput (cmd/s), and Queue Seen On Arrival Statistics From Isolation Measurements: Mean, Coefficient of Variation (cv), 95th Percentile, and Autocorrelation Function Lag 1 (autocorr(1)).

		Measured in Isolation					
Metric	*Statistic*	*File*		*Mail*		*Web*	
		R	W	R	W	R	W
Response Time	mean	20.0	4.82	17.3	2.35	11.8	6.78
	cv	1.78	8.18	1.31	5.20	1.92	6.16
	95th perc.	65.7	9.22	53.2	4.62	35.4	16.1
	autocorr(1)	0.75	0.58	0.54	0.36	0.72	0.69
Throughput	mean	330	238	245	370	470	44.3
Arrival Queue	mean	9.67	8.47	7.86	8.32	8.35	9.41

4.6.3 Model Validation

In this section we present measurement results of heterogeneous workload consolidations, as well as the prediction quality of the proposed linear estimators and the simulation model. For completeness the result set is additionally complemented with homogeneous consolidation scenarios. Validation results are reported as:

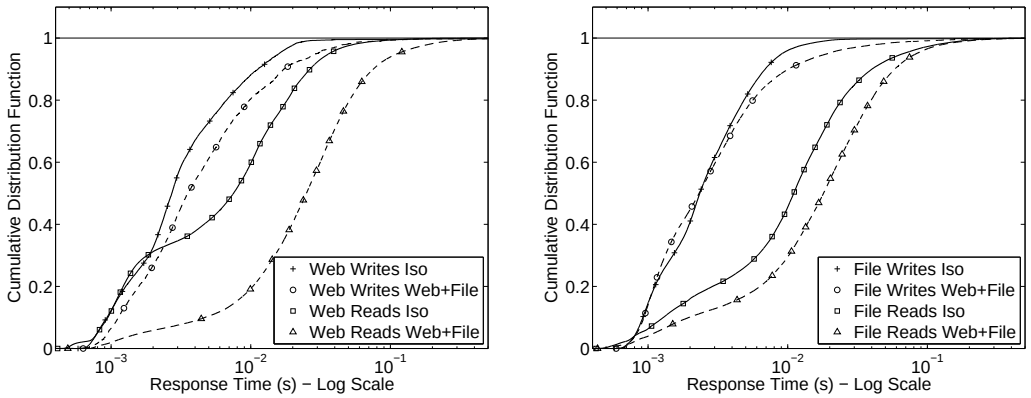
$$\Delta = \left| \frac{\text{measurement} - \text{prediction}}{\text{measurement}} \right|, \tag{4.12}$$

which is the absolute relative error of predictions compared to measurement. We consider approximations with absolute relative error values < 0.30 as being of sufficiently good accuracy. Linear estimation formulas deliver predictions of mean read/write mixes, throughputs, and response times. The simulation model outputs per request response times.

4.6.3.1 Measurement Results

We compare response time and throughput measurements for read/write disk I/O requests obtained with the *blktrace* tool in isolation and consolidation benchmark experiments. Table 4.6 reveals that we record larger latency averages for read requests compared to write requests across all workloads in isolation. Interestingly, response times of write requests display increased variability with CVs of reads and writes ranging in $[1.31, 1.78]$ and $[5.20, 8.18]$, respectively.

Measurements of heterogeneous workload consolidations reflect the difficulties in predicting such quantities, as the joint resource usage of workload mixes may result in volatile interference effects on disk requests. Table 4.7 shows that consolidation causes latencies of both read/write request types to increase. For read requests we find the highest rates of performance degrada-



(a) Web Server (b) File Server

Figure 4.14: Measured Response Time of Web and File Server Workloads in Isolation (Iso) and Heterogeneous Consolidation Web+File.

tion in web server workloads: In particular, the consolidation with file server workloads leads to the largest increases of 242% and 437% in the cases of Web+File and Web+File+File, respectively. Furthermore, write requests of the mail server workload appear to be most sensitive to consolidation where we record response time increases of mail writes ranging in $[199\%; 543\%]$.

Variabilities in response time measurements are high and not always intuitive: Mail server read latencies, e.g., are measured at 45.5ms in File+Mail and at 39.0ms in the higher utilized scenario of Mail+Web+Web. We use the example of Web+File to illustrate how diverse consolidation may effect response time distributions. Figure 4.14 (a) reveals that read request response times of the web server are significantly more affected by Web+File consolidation than the read requests originating from the file server depicted in Figure 4.14 (b). Probabilities of larger write request response times are higher in consolidation for both web and file server workloads. However, the web server workload suffers larger write request response time penalties than the file server.

In homogeneous workload consolidations response time increases are not as severe as in heterogeneous ones. We record increased response times of read requests ranging in $[72\%; 125\%]$ and $[157\%; 239\%]$ for two and three VM consolidation scenarios, respectively. The measurements in Table 4.8 reveal that, similar to the heterogeneous consolidations, we record the largest degradation of read request latencies for workloads of the web server. In the case of write requests response time increases for two and three VM scenarios range in $[17\%; 170\%]$ and $[86\%; 385\%]$, respectively.

The sharing of the storage resource causes I/O throughputs of both request types to drop. Workload interference effects are clearly visible, e.g., in the case of Web+Mail where the read throughput of the mail server workload is monitored at 132cmd/s whereas we only record

Table 4.7: Measurement of Throughputs and Response Times for Heterogeneous Workload Consolidation.

	Measured Response Times (ms)						***Measured Throughputs (cmd/s)***					
Workload	*VM1*		*VM2*		*VM3*		*VM1*		*VM2*		*VM3*	
	R	W	R	W	R	W	R	W	R	W	R	W
File+Mail	35.0	6.90	45.5	7.92	N/A	N/A	198	138	95	190	N/A	N/A
Web+File	40.3	9.45	29.6	7.19	N/A	N/A	225	18	223	158	N/A	N/A
Web+Mail	33.0	7.97	28.0	7.03	N/A	N/A	250	25	132	256	N/A	N/A
Mail+Web+Mail	46.8	12.3	55.5	12.0	45.0	13.0	78	155	145	24	77	167
Mail+Web+Web	39.0	13.5	47.3	12.0	48.5	13.3	86	159	175	27	177	22
Web+Web+File	55.2	15.5	54.0	16.1	43.6	16.7	174	16	173	20	148	96
Web+File+File	63.4	19.1	49.8	19.5	51.0	19.0	155	16	125	91	133	88
Mail+Mail+File	54.9	14.6	54.9	15.1	45.4	13.1	81	147	78	153	146	96

Table 4.8: Measurement of Throughputs and Response Times for Homogeneous Workload Consolidation.

	Measured Response Times (ms)						***Measured Throughputs (cmd/s)***					
Workload	*VM1*		*VM2*		*VM3*		*VM1*		*VM2*		*VM3*	
	R	W	R	W	R	W	R	W	R	W	R	W
File+File	35.4	7.64	34.3	7.60	N/A	N/A	173	108	186	122	N/A	N/A
Mail+Mail	38.5	4.76	37.1	6.35	N/A	N/A	117	208	103	213	N/A	N/A
Web+Web	26.6	8.19	25.0	7.93	N/A	N/A	300	25	277	29	N/A	N/A
File+File+File	52.9	18.1	51.3	15.7	53.0	15.0	116	70	109	70	121	72
Mail+Mail+Mail	52.7	9.79	53.3	11.4	52.0	10.1	71	137	64	152	72	158
Web+Web+Web	40.0	12.6	38.8	13.0	39.4	13.2	204	23	210	27	214	23

95cmd/s throughputs for mail server read requests in File+Mail. For maximizing mail server read throughputs consolidating web and mail server workloads appears to be preferable, since the difference in read throughput is only minor for the web server in both Web+Mail and Web+File.

Summarizing, our measurements of homogeneous and heterogeneous workload consolidations show workload interference effects that may defy intuition and further motivate the need for accurate performance predictions that can support workload placement decisions.

4.6.3.2 Prediction Accuracy of Read/Write Mixes

Tables 4.9 and 4.10 report approximation errors of read/write mix predictions from the linear estimators for heterogeneous and homogeneous consolidations, respectively. Prediction results for two VM heterogeneous workload consolidations are excellent with errors ranging in $[0.02, 0.12]$. When moving to higher utilizations where three VMs are consolidated on the server modeling results slightly worsen, but still remain in a low range of $[0.01, 0.20]$. The linear estimators work equally well for the homogeneous workload consolidations. In particular, results for the web server are accurate. In contrast, predicting the mix of mail server consolidations is more difficult and delivers the largest single error of 0.26 in the three VM case. Given the black box view of the system, as well as the simplicity of the modeling approach the quality

Table 4.9: Accuracy of Read/Write Mix Predictions for Heterogeneous Workload Consolidations from Linear Predictors.

	Measured Mix		***Predicted Mix***	
Workload	*Total*		*Accuracy*	
	R	W	Δ_R	Δ_W
File+Mail	0.47	0.53	0.03	0.02
Web+File	0.72	0.28	0.03	0.07
Web+Mail	0.58	0.42	0.09	0.12
Mail+Web+Mail	0.46	0.54	0.20	0.17
Mail+Web+Web	0.68	0.32	0.07	0.16
Web+Web+File	0.79	0.21	0.01	0.04
Web+File+File	0.68	0.32	0.01	0.01
Mail+Mail+File	0.44	0.56	0.05	0.04

Table 4.10: Accuracy of Read/Write Mix Predictions for Homogeneous Workload Consolidations from Linear Predictors.

	Measured Mix		***Predicted Mix***	
Workload	*Total*		*Accuracy*	
	R	W	Δ_R	Δ_W
File+File	0.61	0.39	0.05	0.07
Mail+Mail	0.34	0.66	0.16	0.09
Web+Web	0.91	0.09	0.0007	0.008
File+File+File	0.62	0.38	0.06	0.10
Mail+Mail+Mail	0.32	0.68	0.26	0.12
Web+Web+Web	0.89	0.11	0.02	0.18

of read/write mix predictions is very good.

4.6.3.3 Prediction Accuracy of Throughputs

We show validation results for predictions of the total read/write throughput according to the linear estimator (4.3), as well as per-VM throughput estimates. Tables 4.11 and 4.12 illustrate that approximation errors for total throughputs are equally low as when predicting read/write mixes. Here results for two VM and three VM heterogeneous cases are similar and range in $[0.02, 0.13]$ and $[0.02, 0.20]$ for throughputs of read and write requests, respectively. Except for the 0.35 value in the three VM consolidation of web server workloads, our method delivers prediction errors of similar quality to homogeneous consolidations.

We are especially interested in the accuracy of per-VM throughput predictions. The decomposed simulation model leverages these estimates to approximate request class populations and so large errors for per-VM throughputs may also reflect on response time predictions of the simulation model. Our approach delivers mostly fair results < 0.30 for reads in hetero-

Table 4.11: Accuracy of Throughput Predictions for Heterogeneous Workload Consolidations from Linear Predictors.

	Predicted Total		***Predicted per-VM***					
Workload	*Total*		*VM1*		*VM2*		*VM3*	
	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W
File+Mail	0.02	0.02	0.22	0.19	0.39	0.10	N/A	N/A
Web+File	0.11	0.16	0.01	0.15	0.21	0.20	N/A	N/A
Web+Mail	0.09	0.15	0.16	0.20	0.04	0.34	N/A	N/A
Mail+Web+Mail	0.08	0.19	0.07	0.19	0.04	0.37	0.15	0.16
Mail+Web+Web	0.07	0.20	0.03	0.16	0.09	0.41	0.10	0.27
Web+Web+File	0.13	0.14	0.15	0.12	0.04	0.18	0.21	0.13
Web+File+File	0.09	0.08	0.06	0.11	0.08	0.09	0.13	0.06
Mail+Mail+File	0.10	0.15	0.02	0.15	0.11	0.09	0.28	0.22

Table 4.12: Accuracy of Throughput Predictions for Homogeneous Workload Consolidations from Linear Predictors.

	Predicted Total		***Predicted per-VM***					
Workload	*Total*		*VM1*		*VM2*		*VM3*	
	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W
File+File	0.07	0.03	0.05	0.09	0.10	0.01	N/A	N/A
Mail+Mail	0.12	0.09	0.03	0.13	0.23	0.06	N/A	N/A
Web+Web	0.16	0.14	0.24	0.15	0.07	0.15	N/A	N/A
File+File+File	0.04	0.12	0.06	0.12	0.03	0.15	0.08	0.11
Mail+Mail+Mail	0.20	0.14	0.12	0.12	0.32	0.13	0.16	0.16
Web+Web+Web	0.22	0.35	0.26	0.39	0.19	0.38	0.21	0.28

geneous consolidations, with a single outlier of 0.39 for File+Mail. Interestingly, our model appears to capture three VM cases better than scenarios with two VMs. Approximation errors for writes range in $[0.06, 0.41]$.

Similarly, prediction errors for reads are low in homogeneous workload consolidations and range in $[0.03; 0.32]$. The accuracies of per-VM write request throughput predictions are slightly worse, but still of fair quality: Errors mostly are ≤ 0.15, except for the web server workload where we find the single largest error value of 0.39.

Notice that read requests have errors generally lower than write requests; this is a positive property since predicting the performance of read requests, which are mostly synchronous, is much more relevant than predicting the performance of write requests, which are mostly asynchronous and thus do not impact much on the response times of the applications in the VMs.

Table 4.13: Accuracy of Response Time Predictions for Heterogeneous Workload Consolidations from Linear Predictors.

	Predicted per-VM					
Workload	*VM1*		*VM2*		*VM3*	
	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W
File+Mail	0.01	0.10	0.23	0.07	N/A	N/A
Web+File	0.25	0.16	0.04	0.50	N/A	N/A
Web+Mail	0.19	0.20	0.02	0.30	N/A	N/A
Mail+Web+Mail	0.15	0.10	0.30	0.03	0.12	0.14
Mail+Web+Web	0.17	0.07	0.33	0.27	0.35	0.11
Web+Web+File	0.32	0.09	0.31	0.05	0.12	0.002
Web+File+File	0.24	0.21	0.02	0.24	0.04	0.21
Mail+Mail+File	0.08	0.36	0.08	0.38	0.11	0.27

Table 4.14: Accuracy of Response Time Predictions for Homogeneous Workload Consolidations from Linear Predictors.

	Predicted per-VM					
Workload	*VM1*		*VM2*		*VM3*	
	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W
File+File	0.08	0.18	0.11	0.18	N/A	N/A
Mail+Mail	0.17	0.08	0.14	0.18	N/A	N/A
Web+Web	0.27	0.56	0.23	0.61	N/A	N/A
File+File+File	0.07	0.28	0.10	0.16	0.06	0.12
Mail+Mail+Mail	0.11	0.19	0.12	0.30	0.09	0.22
Web+Web+Web	0.33	0.49	0.31	0.45	0.32	0.43

4.6.3.4 Prediction Accuracy of Response Times

Linear Estimators. The heterogeneous workload consolidation results in Table 4.13 indicate that our heuristic for response time predictions works well for read requests, which are the most important to predict; most of absolute relative errors are in a narrow interval ranging in $[0.01, 0.25]$, while a few cases show larger errors ranging in $[0.31; 0.35]$. In particular, consolidation scenarios of web server workloads appear to be difficult to predict for the linear estimators. However, in light of the above addressed 435% measured latency increase for consolidated web server workloads, such prediction errors may be satisfactory. The situation is similar for homogeneous workload consolidations, where Table 4.14 further confirms that response time predictions for the web server are the most difficult. Approximation errors for mail and file server workloads range in $[0.06; 0.17]$.

Results for write request response times contain some larger error values and are in a few cases significantly worse than the read results. In heterogeneous consolidations results range in $[0.002; 0.38]$, apart from the difficult case of Web+File. Our records of homogeneous consoli-

Table 4.15: Accuracy of Response Time Predictions for Heterogeneous Workload Consolidations from Simulation.

	Predicted per-VM					
Workload	*VM1*		*VM2*		*VM3*	
	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W
File+Mail	0.17	0.33	0.10	0.20	N/A	N/A
Web+File	0.09	0.67	0.25	0.17	N/A	N/A
Web+Mail	0.11	0.36	0.49	0.28	N/A	N/A
Mail+Web+Mail	0.06	0.49	0.21	0.18	0.04	0.46
Mail+Web+Web	0.14	0.50	0.26	0.14	0.19	0.23
Web+Web+File	0.29	0.35	0.37	0.33	0.21	0.67
Web+File+File	0.33	0.38	0.25	0.60	0.26	0.60
Mail+Mail+File	0.12	0.51	0.16	0.51	0.06	0.58

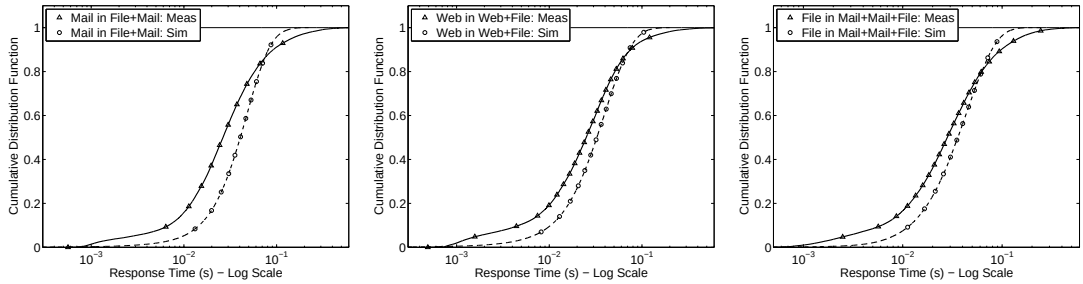
Table 4.16: Accuracy of Response Time Predictions for Homogeneous Workload Consolidations from Simulation.

	Predicted per-VM					
Workload	*VM1*		*VM2*		*VM3*	
	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W
File+File	0.07	0.51	0.14	0.31	N/A	N/A
Mail+Mail	0.24	0.47	0.24	0.14	N/A	N/A
Web+Web	0.29	0.28	0.17	0.35	N/A	N/A
File+File+File	0.30	0.44	0.21	0.47	0.24	0.47
Mail+Mail+Mail	0.02	0.32	0.07	0.39	0.12	0.35
Web+Web+Web	0.12	0.20	0.22	0.18	0.13	0.18

dations show mostly small error values ≤ 0.30. Once again the web server workload produces some large error value outliers. However, mean relative errors over all results compare at 0.17 and 0.23 for reads and writes, respectively.

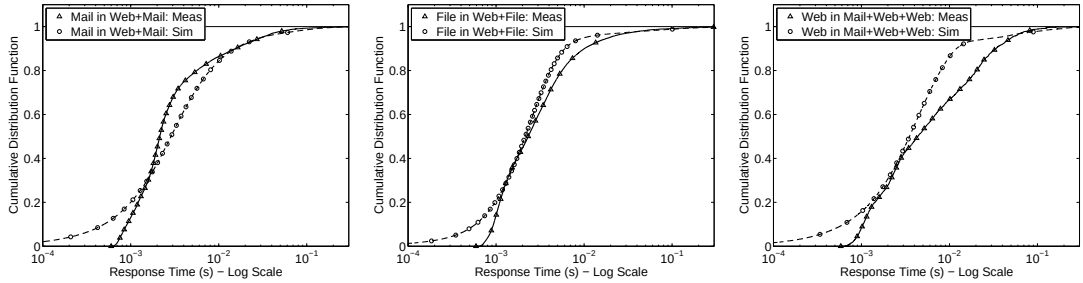
Simulation Model. We focus our validation of simulation results on the performance critical read requests. Tables 4.15 and 4.16 illustrate that approximation errors of mean response times are in line with approximation errors of the analytical solution. For heterogeneous consolidations read request error values range mostly in $[0.02; 0.33]$. Additionally, we record two larger errors of 0.49 and 0.37 in Web+Mail and Web+Web+File, respectively. Predictions for reads are more accurate in homogeneous consolidation, where the single largest error value yields 0.30.

Response time predictions of write requests are more difficult due to high variability in service times with $CV_{a_k}^W = [1.03; 13.5]$ and some cases where read requests interfere with writes as depicted in Figures 4.4 (b), (d), and (f). Our simulation model approximates the high variability in write request service demands with a Hyper-exponential distribution. Predictions of write request response times have mean relative errors of 0.34 and 0.40 for two VM and three



(a) File+Mail (b) Web+File (c) Mail+Mail+File

Figure 4.15: Distributions of Measured and Simulated Read Response Times in Consolidation.



(a) Web+Mail (b) Web+File (c) Mail+Web+Web

Figure 4.16: Distributions of Measured and Simulated Write Response Times in Consolidation.

VM consolidations, respectively. Some prediction errors are large with values > 0.50.

We choose an example case for each workload type to illustrate the potential of our model in fitting simulation results to measured distributions. Figure 4.15 (a) shows that predicted response times in File+Mail are pessimistic across most of the distribution for mail server read requests. The quality of results is equally good for the web server workload in Figure 4.15 (b). We record 90th percentiles of roughly $0.0755s$ and $0.0719s$ for measurement and simulation, respectively. In the higher utilized and more difficult to predict three VM consolidation scenario illustrated in Figure 4.15 (c), the response time distribution of file server requests is also approximated well.

Figure 4.16 (a) illustrates that the simulation model is able to match some measured response time distributions of write requests well. For the mail server workload in Web+Mail the model delivers 90th percentiles of $0.0157s$ and $0.0154s$ for measurement and simulation, respectively. The examples in Figures 4.16 (b) and (c) highlight one of the problems of applying the decomposition approach to write requests. In consolidation the workload interference of write requests originates not only from other writes, but also from reads. The decomposition model does not account for these additional overheads and leads to optimistic write request response time predictions.

Summary. The validation confirms that the proposed approximations could be effective in several cases of practical interest. Predictions of read/write mixes and request throughputs are of high quality. Approximating read request response times is more difficult, but again our models deliver fair results. More work is needed towards investigating the behavior of write requests, which appear to be the harder to predict in consolidation. The often asynchronous nature of write requests [171] may be more accurately captured with an open queueing model instead of our closed simulator. However, the capability of our models to capture read request performance more accurately than write request performance is a positive property as read workloads are mostly synchronous and thus may directly affect the performance of applications.

4.7 Summary and Conclusions

Conclusions. We have presented simple models that can predict response times, throughputs, and read/write mixes of disk I/O requests when consolidating workloads on to a shared storage device. Our contributions are threefold. Firstly, we parameterize the model with in-VM measurement data only instead of instrumenting the VMM. Secondly, we define simple linear estimators that decompose the workload into read and write requests for prediction of mean performance metrics in heterogeneous workload consolidations. Thirdly, we complement the decomposed modeling approach with a simulation model that approximates response time distributions of consolidated workloads.

Proposed methodologies are validated against system measurements. Validation experiments with emulated application workloads show that the decomposition model can accurately predict read/write mixes and throughputs for a variety of heterogeneous and homogeneous consolidation scenarios. Response times are more difficult to predict, but the model delivers fair results for predictions of read request response time means and distributions. For write requests we record some large error values resulting from the decomposition property being less evident for writes. A positive feature of our models is that approximation results are generally better for read requests which are the more relevant in practice. Read requests are mostly synchronous and cause the submitting application threads to block. Thus the performance of read requests may directly affect application performance.

Limitations. Our approach is limited to scenarios where fine-grained monitoring data is available for a VM workload configuration. In practice such detailed monitoring may not always be available or might be considered invasive. While our simple models work well in the presented environment, specialized layered queueing models may be beneficial when modeling more complex cloud topologies [117, 132].

Usability. We envisage our methodologies being used by system administrators of cloud environments. The disk I/O performance prediction obtained from our models can support application placement at VM granularity, e.g, for pre-configured VM images of file and mail servers. It requires the availability of a trace repository consisting of traces collected from VM images running in isolation on the virtualized server.

Part II

Database Resource Management

5 Performance Evaluation of Query Admission Policies for In-Memory Database Systems

5.1 Introduction

The availability of system memory at low cost in combination with the parallelization capabilities of multicore servers have prompted in recent years the adoption of in-memory database technology in enterprises [44, 176]. The workload of in-memory databases often consists of long running batch jobs which process very large data sets of customer information and sales records. Since longer execution times often correspond to increased costs, in-memory databases have emerged as a mainstream technology to maximize responsiveness of data-intensive applications [86]. By manipulating large data sets directly in memory, and therefore omitting time-consuming disk operations, in-memory databases execute data-intensive operations in a fraction of the time of traditional databases.

This chapter is motivated by the problem of defining effective resource management methodologies for in-memory database systems. We progress our performance evaluation of data access operations and investigate data sets residing in system memory. In particular, we consider database workloads where multiple queries are running at the same time, i.e. in consolidation. Our main contribution lies in the evaluation of admission control policies for query execution. We provide strong evidence using trace-driven simulation that our policies are more efficient than popular policies such as first-come first-served (FCFS) or shortest expected processing time first (SEPT).

Database systems essentially process queries in two steps. First the query text is parsed into an internal representation referred to as an *operator tree*. Secondly, the internal query representation has to be translated into a procedural program executable by the database query engine denoted as a *query execution plan* [136]. A decisive factor for database performance is the maximum number of queries that are allowed to run concurrently in the system [142, 178]. For example, choosing a small concurrency level results in underutilized resources and low throughputs. Conversely, large concurrency levels may induce excessive contention delays for cores or exhaustion of memory resources.

A challenge that arises in defining a good admission control policy for database systems is the interplay between queries that are running at the same time. Related work has highlighted the tendency of queries to affect each others performance when concurrently processed in a database system [39,60]. The interplay effects are not limited to heightened resource contention at the CPU cores, but may also involve delays due to memory access or due to the internal characteristics of the software such as limited application threads.

The observed performance may not only depend on the total number of queries executing concurrently, but also on the specific query mix in service. A query mix often comprises queries of varying types, i.e. classes. For example, the decision support benchmark TPC-H defines 22 query classes with varying *SQL* expressions, e.g., consisting of distinct SELECT, GROUP BY, and ORDER BY statements. Query completion times may decrease when queries share data loaded into buffers. Conversely, completion times are impacted negatively if queries compete for software resources, e.g, limited thread pools. We define the performance interplay effects not directly related to CPU core contention as query *interference*. An important factor determining the interference inherent in the running mix is the extent of backlogs of jobs in the system resulting from the task parallelization introduced by the query planner. The query plan is organized as a sequence of consecutive or parallel subtasks that are spawned synchronously, thus suddenly increasing the number of jobs backlogged in the system. Query plans usually vary for queries of different types resulting in different parallelization levels determined at run-time and which may also change during the lifetime of the query.

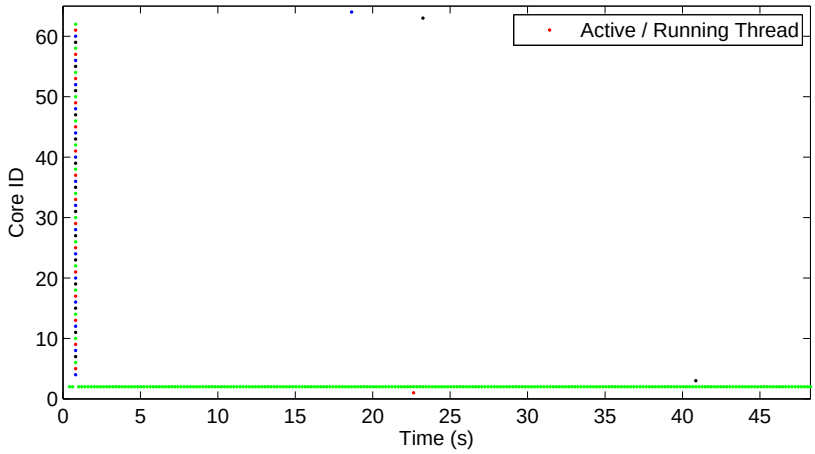
As an example illustration of parallelization levels in database workloads Figures 5.1(a)-(b) show CPU thread affinity traces collected on our reference system, introduced in Section 5.6.1. In particular, we run a single query instance and monitor over time the system cores observed to execute threads belonging to the query process. The example trace in Figure 5.1(a) conveys a query type with a mostly synchronous execution pattern as the query planner utilizes only a single core for large portions of the query processing. System resources appear to be utilized more evenly in the example depicted in Figure 5.1(b), where the query plan entails more concurrency for this particular type of query.

This chapter proposes novel admission control policies for database systems. Our policies aim to make efficient use of system resources and are derived solely from system measurements. The database application is treated as a black box. Summarizing we make four contributions:

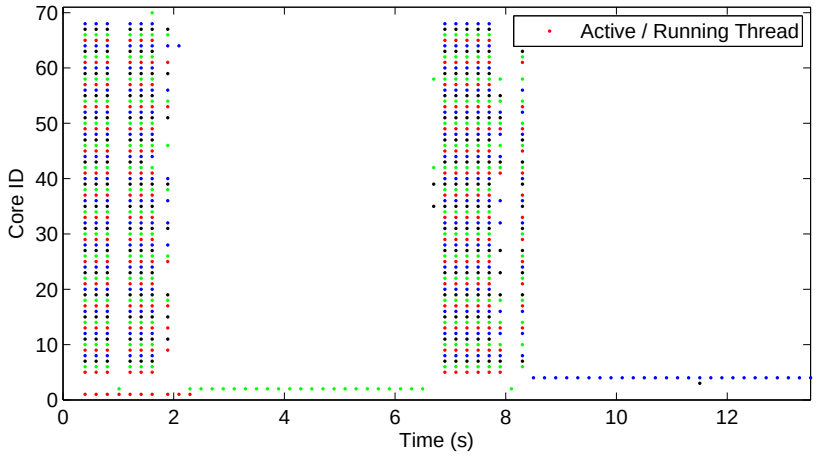
1. We propose admission control policies derived from (i) mean job queue lengths, (ii) CPU utilization, and (iii) a no-interference baseline measure for query class pairs computed from ratios of queue lengths and utilization. The policies ignore I/O subsystems and therefore are highly applicable to in-memory databases. In essence, our policies maintain a table that characterizes the interference and prioritize query classes that are able to use resources more efficiently, typically at the expense of other query classes.

2. For evaluation of admission control policies we develop a trace-driven simulation model. The closed queueing model is validated against system measurements.

3. Using the trace-driven simulation model and simplified synthetic workloads we present a comparative evaluation of the proposed admission control policies. In particular, we com-



(a) Example Thread Affinity Trace with Low Parallelization Levels



(b) Example Thread Affinity Trace with High Parallelization Levels

Figure 5.1: Example CPU Thread Affinity Traces Measured During Single Query Executions. Each Measurement Point Accounts for a System Core Running a Thread at Sample Time.

pare throughput, mean response time, and mean weighted slowdown results to the established policies FCFS and SEPT.

4. We further validate the queue length and no-interference baseline policies using TPC-H [12] type workloads and system traces from a commercial solution, SAP HANA [86], a product that aims at improving the performance of complex business intelligence workloads using in-memory database technology.

Although our work is illustrated on a specific product, the proposed methodologies are general and may be applied to other in-memory database solutions. The remainder of the chapter is organized as follows. Section 5.2 gives an overview of related work and Section 5.3 states the addressed research questions. The proposed resource management methodologies are intro-

duced in Section 5.4. In Section 5.5 we present the trace-driven simulation model and validate its quality in Section 5.6. The resource management methodologies are evaluated in Section 5.7 and Section 5.8 offers a summary and conclusions.

5.2 Related Work

This section summarizes related work and is structured into the research areas scheduling and admission control, load balancing, query interaction, and simulation models.

5.2.1 Scheduling and Admission Control

There is a long history of studying scheduling algorithms for parallel processing [128]. In the context of databases, related work can be categorized into approaches prioritizing request classes (i) inside or (ii) outside of the database system. Researchers initially focused on internal scheduling, i.e., prioritizing transactions as they are run inside the database. The work in [112,120] introduces internal schedulers for the correct coordination of concurrent read/write requests. In [64] priority-based schemes for management of DBMS resources, e.g., disks and memory buffers are presented. Transaction scheduling and concurrency control in the presence of time constraints have been studied for real-time database systems [29–31]. In [140] the bottleneck resources CPU, synchronous I/O, and table locks are identified and subsequently used for derivation of internal prioritization policies. The same authors later provide a statistical analysis of locking with OLTP workloads and evaluate preemptive and non-preemptive prioritization techniques [141].

More recently and closer to our work, external or interposed schedulers have been proposed. External scheduling is typically implemented using admission control in order to prioritize transactions before they enter the database system [140]. These techniques maintain black box views of the applications and govern the order in which requests are admitted to the system through an external queue. Compared to internal schedulers, this method is less complex to implement and allows more flexible development of custom scheduling policies as database internals do not need to be considered.

External schedulers have been proposed for a range of systems and application scenarios, e.g., for the fair and proportional sharing of storage utilities [114] and admission control for web servers [84]. For example, the approaches in [54, 197] introduce admission queues and web server request prioritizations in an effort to provide quality-of-service (QoS) goals under high system load. While some admission control policies for web servers advocate dropping of requests during overload phases [54], our approach focuses on database systems and does not allow request dropping. Furthermore, the comparative evaluation of our policies focuses on the

performance metrics throughput, response time, and slowdown and leaves an analysis of the fairness as a possible extension. However, the simulation results provided in Figures 5.13 (a-c) show that the probabilities for extremely large response times, e.g. due to request starvation, are lower using our policies compared to the prioritization of the expected shortest running queries.

A large body of admission control research for databases addresses the computation and configuration of the multi-programming limit (MPL). This quantity specifies the total number of transactions admitted to the database at any point in time, i.e. the number of transactions concurrently in service. Setting the MPL too large may result in performance degradation due to memory and data contention, commonly referred to as *thrashing* [80]. On the contrary, low configuration of MPLs can lead to underutilized system resources and insufficient throughput. Related work provides techniques for estimation of maximal [65, 109, 151] and minimal [178] MPLs. Instead, we consider the MPL as an optimally configured and available input parameter and focus on the order of query admission to the database, i.e., request prioritization.

A common way to classify prioritization techniques is to distinguish between preemptive and non-preemptive disciplines [75]. Since we consider admission control policies external to the database system without the possibility of interrupting running requests inside of the database, this work focuses on non-preemptive disciplines. Non-preemptive approaches may be further categorized into non-size based and size based disciplines [205]. Recent work has identified the non-size based FCFS as a popular policy in database systems and highlights the sensitivity of its performance to the request arrival order [39]: In FCFS, the performance of short running requests can be negatively affected by queueing delays resulting from wait times for the completion of long running requests.

In size-based techniques the size of requests is assumed to be known at the time of arrival to the queue. For example, in [84] an external admission control for e-commerce web traffic with SJF policy is presented. Contrary to SJF, SEPT [149] at the time of admission is only aware of each request's class. Each class has an associated size expectation and SEPT unconditionally prioritizes the oldest arrived request with the smallest expectation [68]. Queueing theory has shown that, in single server systems, policies favoring short tasks provide lower mean response and waiting times than non-size based policies [75, 78]. In addition, SJF is known to maximize global system throughputs. However, prioritization of short running jobs at the expensive of long running jobs may lead to the starvation of low priority requests. Related database research has shown that in the presence of hardware, software, and data access contention, SJF may perform worse than FCFS as admission control policy for batched workloads [37]. In this work we use the established disciplines FCFS and SEPT as a baseline for comparison of our proposed admission control policies.

Furthermore, the work in [177] presents the *External Queue Management System (EQMS)*

for governance of QoS targets with multiclass database workloads. The size-based policy prioritizes request classes in order to meet per-class goals of response time mean, variance, and percentiles. Prioritizations are derived as a function of predefined goals and historical response time measurements. Similarly, *Query Patroller* [23] is a commercial tool for IBM DB2 databases facilitating external admission control and static query prioritization, e.g., for short running over long running ones. None of these techniques specifically considers the parallelization of requests into subtasks in the scheduling decisions.

5.2.2 Load Balancing

In systems with multiple processing entities, load sharing can be critical in achieving favorable throughput and response time performance [107]. The incoming requests typically enter a router or dispatcher before they are assigned to a processor according to a load balancing strategy. A large number of load balancing strategies have been proposed in the literature, e.g., for distributed database systems [208]. Queueing theory may be employed to derive system performance estimates and optimal deterministic or probabilistic allocations of tasks to servers.

The work in [57] presents a simulation study of load balancing policies that base assignment decisions on queue lengths at the servers and finds that the join-the-shortest-queue (JSQ) [111, 206] policy minimizes the mean response time for servers with PS scheduling and exponential service. The JSQ policy forwards an incoming request to the server with the least number of unfinished requests. More recently, in [106] an analytical approach for approximation of the steady-state queue length distribution in a network with JSQ routing, PS servers, and general service time distribution is introduced. However, with high variability in service time distributions JSQ is no longer an optimal strategy [204]. The technique in [107] specifically addresses heavy-tailed service time distributions by assigning job time limits: In case a job does not complete within the time limit it is killed and restarted elsewhere.

A large number of the classical load balancing literature is concerned with single-class workloads [156]. In a more recent work a generalization of JSQ policies with multi-class queues is provided [43]. The preemptive approach considers $M/G/1$ queues and routes jobs to the least congested station, where the measure of congestion is linear in the per class queue lengths at the stations. Similarly, the proposed join-minimum-cost-queue (JMCQ) policy generalizes JSQ for multi-class workloads using a cost function based on the per queue join price and queue length [187]. In contrast to the previous techniques our queue length based policies are not aware of the state of the individual queues internal to the database application. Instead, we prioritize the class of requests that on average impose the smallest increases in joint queue length at the database.

5.2.3 Query Interaction

The typical workload of database systems consists of a number queries of multiple types, i.e. classes. The performance of concurrently running queries can be impacted by interference effects, e.g., if two queries compete for CPU resources or share data loaded into a buffer [37]. Multiple Query Optimization (MQO) techniques account for job interference by exploiting database application implementation details and query structures to optimize query plans [163, 182]. In particular, MQO techniques address the problem of optimizing sets of queries which may have common sub-expressions [170].

Closer to our work are the following methods maintaining black box views of the database. The authors in [36–40] investigate interactions between batched TPC-H queries and propose the *QShuffler* algorithm for interaction-aware scheduling based on experimental sampling and regression techniques. The approach considers only the total time to completion of a query batch.

Other work studies query pairs for quantification of workload interference. The approach in [41] uses a similar computational model to QShuffler, but expresses the response time of a concurrently running query mix as a linear combination of execution time shares of the individual query class pairs inherent in the mix. Queries are first classified based on their unique SQL skeletons. During a training phase the influence that the concurrent execution of a query pair of the two classes c and r has on the mix is studied. In [157] the authors rely on measurements of disk accesses for sequential and concurrent execution of TPC-R [13] queries to determine interference of query classes. The monitoring approach aims to exploit shared buffers and caching effects. We instead use a combination of queue length and CPU measurements to build our model.

5.2.4 Simulation Models

Simulation models for performance evaluation of database designs have been proposed as early as the ’70s [154], but did not prevail as a commonly used tool of database administrators. In particular, the authors derive an event-driven simulation model for performance evaluation of the operating system, database, and applications prior to development. In [59, 64, 66] simulation models are employed for evaluation of internal database resource schedulers. The database system is modelled as a collection of files representing relations, while CPU and disk resources are represented by queues. The approach in [192] uses a probabilistic cost model to parameterize a simulation model for analysis of transaction scheduling in distributed real-time databases. In [159] simulation is applied to generate database I/O traces.

Closer to our approach, the authors in [41] mention a simulation framework for evaluation of

query scheduling disciplines. The simulation model captures the number of processor and disk cycles the database can perform per second, and the available main memory. Query classes are parameterized with the number of required processor cycles, disk operations, and memory. The framework is not validated against a real system. More recently, a simple mathematical simulation model for batched queries is presented in [40]. None of these tools consider the parallelization of queries into subtasks or are parameterized from system traces. Instead of capturing detailed database internals, our approach aims to model accurately the query performance metrics response time, throughput, and queue lengths.

5.3 Research Questions

The work in this chapter addresses the following research questions in the area of in-memory database systems:

1. Can admission control policies based on system measurements improve query performance compared to the established policies FCFS and SEPT?
2. How does the performance of policies based on system measurements of queue lengths, CPU utilization, and query class pairs relate?
3. Should admission control polices derived from mean queue lengths be based on arrival instant or time averaged queues?
4. Does a query no-interference baseline derived from query class pairs produce better results compared to the policies with less overheads, i.e. mean queue lengths and CPU utilization?

5.4 Resource Management Methodologies

In this section we propose four admission control policies for in-memory database systems. A query that runs on a database has to compete with concurrently executing queries for core, memory, and software resources. Our admission policies aim to improve the system performance via prioritization of workload mixes making efficient use of the available resources. The choice of query admission order is complicated by high variations of parallelization levels and service demands inherent in the workload. In particular, our proposed policies derive request prioritizations from measured system queue lengths, CPU utilizations, and a query-pair based no-interference baseline measure. We specifically choose policies based on single class measurements and pairwise class comparisons for our evaluation. While pairwise approaches have been introduced by related work [41], single class measurements are easier to maintain and impose less overheads on the admission controller.

5.4.1 Queue Length Policies

The queue length policies take into account the average number of additional tasks, i.e. threads, imposed on the system by the admission of a request. In particular, we distinguish between two approaches considering the (i) mean arrival instant queue length and (ii) the time averaged queue length. Let A_r^c be the mean number of tasks of class r in execution at the database as observed by a query of class c at the instant of admission to the database, including the newly arrived task if $r = c$. The arrival queue length policy prioritizes the class r that on average adds the smallest amount of backlog to the system conditional on the arrival instant at the database. For example, if the number of classes is $K = 2$ and the mean arrival queue lengths observed are $A_r^c = 2$ and $A_c^r = 3$ then the admission policy will assign higher priority for requests of class r. This metric has the advantage of accounting indirectly for the response time of each class. It is well known in queueing theory that, in the absence of prioritization mechanisms, the arrival queues A_r^c are the main determinants of end-to-end response times [130]. Therefore, this metric accounts also for the time each request spends in the system.

The second queue length based admission policy builds on the same methodology, but utilizes the time averaged queue lengths. This metric has the advantage of fewer overheads as it associates only a single value with each request class r unconditional of any other class c. We denote the long time queue length average Q_r.

5.4.2 Utilization Policy

The CPU processing capacity is an important system resource for the performance of in-memory database systems. Our utilization policy aims to make efficient use of the processors by prioritizing the most CPU intensive request classes. The processing demands of database requests can exhibit high variations even for requests with identical end-to-end response times. For example, two requests of different classes may both run for $10s$, but on average respectively utilize 1 and 64 cores in parallel. See Figures 5.1(a-b) for an illustration of the variance in parallelization levels observed in the reference workload. Let U_r be the mean system utilization of class r queries averaged across cores. The admission policy prioritizes the classes with highest U_r values at the expense of requests which utilize the CPUs less.

5.4.3 Pairwise Query Interference Policy

Performance modeling theory has developed over the years analytical tools to describe contention for core access using stochastic processes, e.g., product-form queueing network models [49]. This theory explains resource contention between different jobs when these are served under processor-sharing scheduling, an idealized version of round-robin where the quantum of

time that each job receives is infinitesimally small. Processor sharing scheduling is well known to idealize CPU scheduling [57]. Product-form queueing network models are an appropriate abstraction to describe core contention, but they place some limitations on the analysis of real-world systems. For example, queries submitted to a database server may also experience blocking delays caused by constraints on the maximum software threading level. Thus, a query class with high parallelism level, but low core processing requirements, may still block the execution of other query classes if it saturates the threading limit constraint. These blocking delays add to the core contention overheads and can be difficult to model. We shall refer to such delays as *interferences* and, with the aim of defining a new admission control policy, we aim to characterize and exploit interferences between query classes.

We present in this section two contributions. First, we define a metric for characterizing the interference between any pair of query classes. Next, we propose an admission control policy to leverage this information for better resource management for in-memory databases. Specifically, we follow this approach:

1. We first define a simplified queueing model for the performance of the database server in the idealized case where there is core contention, but *no* interference, between query classes. Thus, this system ignores aspects such as task forking and admission control. We prove that, in this idealized model, a certain relationship must exist between utilization and queue-lengths (i.e., backlogs of queries) seen at the cores at the instant of admission into the system of a new query by the planner. We shall refer to this relationship as the *no-interference baseline*.

2. We then focus back on the real system. Based on the idealized result, we define a metric that quantifies how much the behavior of queries in the real system deviates from the no-interference baseline. This will be used as a metric for admission control.

3. Finally, we define an admission control policy that tries to prioritize query classes that are expected to maximize core utilization in the presence of interference.

5.4.3.1 No-Interference Baseline

We begin by defining an idealized model for the database server performance by considering a closed queueing network composed by M processor-sharing queues, modelling the cores, and a delay station, modelling the time taken by the scheduler to issue the next query after completion of the previous one. This time, called *think time*, is zero if the scheduler queue contains jobs, otherwise equals the residual time before a client issues the next request to the database. We denote by Z_c the mean think time of a class-c query. We ignore in this idealized model aspects such as maximum threading levels, thread forking, and latency for memory access. In order to

mimic the behavior of threads, we require each job to perform on average a number of visits to each core, before completing, such that the total processing demand $D_{c,i}$ of class-c queries, summed across visits, placed on each core i is identical to the one in the real system for each query class. Since cores are assumed identical, we define $D_c = D_{c,1} = D_{c,2} = \ldots = D_{c,M}$, where M is the number of cores. Let U_c be the utilization of class-c queries averaged across cores. Also, let A_r^c be the mean number of jobs of class r executing at a core, averaged across the cores, as observed by a query of class c at the instant it is issued by the query planner. We prove the following result.[1]

Theorem 1. *Under the above assumptions, utilizations and mean arrival queue-lengths satisfy the following relation*

$$\frac{U_r}{U_c} = \frac{A_r^c}{A_c^r}$$

for all classes $c, r = 1, \ldots, K$ *where* $c \neq r$.

Proof. Under the given assumptions, the model under study is a product-form closed queueing network [49]. The population of the model corresponds to the population of cycling jobs. Let G be the normalizing constant of the equilibrium state probabilities of the underlying Markov chain model that describes the stochastic evolution of the network. Also let the notation $G_{c,r}^+$ indicate the normalizing constant for a related model where we remove a job of class c and a job of class r from the network, but increase the number of cores by one. Let also G_c and G_r be the normalizing constants of models with a job less of class c and r, respectively. For the model under study, it is established that [56]

$$A_r^c = D_r \frac{G_{c,r}^+}{G_c}, \quad A_c^r = D_c \frac{G_{c,r}^+}{G_r}$$

Therefore

$$\frac{A_r^c}{A_c^r} = \frac{D_r}{D_c} \frac{G_r}{G_c}$$

Recall that the class-c and class-r throughputs, respectively denoted X_c and X_r, are defined in terms of normalising constants as

$$X_r = \frac{G_r}{G}, \qquad X_c = \frac{G_c}{G},$$

then

$$\frac{A_r^c}{A_c^r} = \frac{D_r}{D_c} \frac{G_r}{G} \frac{G}{G_c} = \frac{D_r}{D_c} \frac{X_r}{X_c} = \frac{U_r}{U_c}$$

where we used in the last passage that, by the utilization law [130], $U_r = X_r D_r$ and $U_c = X_c D_c$. This completes the proof. □

[1] Theorem 1 and the subsequent proof are contributions by paper co-author Giuliano Casale [125].

Based on the last result, it is natural to define the no-interference baseline as the metric

$$\eta_{r,c} = \frac{U_r}{U_c} - \frac{A_r^c}{A_c^r} \tag{5.1}$$

which for the idealized system satisfies $\eta_{r,c} \equiv 0$ for every possible choice of classes r and c. A discussion of this metric is provided in the next subsection.

5.4.3.2 Interference in the Real System

Using trace-driven simulation and the measurement discussed in the next sections, we tested if the observed utilization levels and arrival queue-lengths in a SAP HANA DB, a real-world in-memory database, satisfied the no-interference baseline. Without loss of generality, the queue-length values for this system are intended to be of threads instead of jobs. Our empirical observation is that, in general, the no-interference constraint is not satisfied, except in the obvious case $r = c$, and different pairs of classes (r, c) exhibit different degrees of deviation from the idealized constraint. Our proposal is to interpret this deviation as a measure of class interference, i.e., the delay occurring between classes that cannot be explained by core contention. We have in particular three cases:

- Case $\eta_{r,c} > 0$. This corresponds to the situation where

  $$\frac{A_c^r}{U_c} > \frac{A_r^c}{U_r}$$

  For example, this may be due to class c placing a higher A_c^r value than expected. Thus, class c consumes the threading limit constraint per unit of utilization *relatively* more than class r. Under this situation, it seems more efficient to execute class r queries instead of class c queries. The same conclusion is obtained after rearranging the terms as

  $$\frac{U_c}{A_c^r} < \frac{U_r}{A_r^c}$$

  which suggests that class r produces more work (higher utilization) per thread queued at the core. Thus, in relative terms, we say that class r is more *work-intensive* than class c per thread queued at the core.

- Case $\eta_{r,c} < 0$. This is similar to the previous case and corresponds to a situation where class c utilizes resources more efficiently with respect to class r per thread queued at the core.

- Case $\eta_{r,c} = 0$. The two classes utilize resources in a fair manner consistent with the no-interference model.

5.4.4 Query Admission Policy Implementation

Based on the observations in the previous subsections, we implement four admission control policies for in-memory databases in a trace-driven simulation model. For a system with K classes, the policies rely on offline definition of a matrix $\boldsymbol{\eta} = [\eta_{r,c}]$, where element $\eta_{r,c}$ is defined by the respective admission control policy for the pair of classes in row r and column c, $r, c = 1, \ldots, K$. The admission policies attempt to optimize system performance by selecting, among the jobs queueing in the system admission queue, the earliest arrived query of class r^* in the admission queue such that

$$\sum_{c=1}^{K} \eta_{r^*,c} N_c^{dec} \geq \sum_{c=1}^{K} \eta_{r,c} N_c^{dec}, \qquad \forall r \neq r^*$$

where N_c^{dec} is the *current* number of class c jobs in service at the cores at the instant when the scheduler needs to take the admission control decision. (Note that $N_c^{dec} \neq A_c^r$ since the latter is instead the *long-term average* for the number of threads observed when a class r query is admitted to service.) In particular, the admission policies we investigate in this study are:

$\eta_{r,c} = -A_r^c$: Chooses to admit the class r job that at the arrival instant of class c jobs on average adds the smallest backlogs to the running mix.

$\eta_{r,c} = -Q_r$: Chooses the class r job that on average adds the smallest backlogs to the running mix.

$\eta_{r,c} = U_r$: Aims to maximize the work intensiveness of the running mix by admitting the query of class r that imposes the largest additional utilization to the running mix .

$\eta_{r,c} = P_{r,c}$: This policy utilizes the pair wise no-interference baseline as defined in(5.1).

5.5 Trace-driven Simulation Model

We defined a simulation model for performance evaluation of the proposed admission control policies for in-memory database systems. The simulation model is helpful for what-if comparison of admission control policies and provides a testbed for computing the $\boldsymbol{\eta}$ matrix defined by our proposed policies. The model implementation extends JINQS [87], a library for simulating multiclass queueing networks, and is parameterized solely from measured system traces. Maintaining a black box view of the database internals is advantageous since such details may not always be available, usually complicate the model, and can require modeling changes once the application implementation is altered.

We consider a workload of clients interactively accessing hosted database services and use the closed queueing model depicted in Figure 5.2 for performance evaluation. The model specifically captures delays due to contention for hardware and software resources. Database performance degradation due to hardware contention has been mainly linked to storage resources in previous work [157]. However, the absence of time consuming disk I/O operations on in-memory platforms allows us to focus on modelling contention delays for cores. We represent available system processors with a network of processor-sharing queues in the model.

In multicore environments, databases can efficiently utilize parallel processors by incorporating concurrency into query plans. Parallelizing large portions of query tasks may result in contention for threads allocated to the application. We model the maximum threading limit with a finite capacity region (FCR) [130]. A FCR is a subnetwork with a constraint on the maximum number of jobs that can simultaneously visit it at a given time. Arriving jobs are added to an *AdmissionQueue* if available capacities are exhausted. No jobs are dropped. The *AdmissionQueue* is governed by a configurable admission control policy, e.g., non-preemptive FCFS, SEPT, etc. When the FCR capacity is exceeded, upon departure of a job from the FCR to the outside, the next job to be admitted is chosen within the *AdmissionQueue* by the admission control policy.

Consider the jth query instance, i.e. job, requesting access to the FCR. In the real system, this corresponds to the jth query received by the database from the clients. In the simulation, we describe this job by a class $k(j)$, which describes the type of operation performed, and by an iteration number $i(j) \leq I(j)$, where $I(j)$ represents the maximum number of cycles job j performs in the FCR before leaving. This corresponds to the number of processing phases the query undergoes before completing. Thus, a cycle corresponds to a single execution round at the cores. We define the response time for job j as the difference between time of its completion at the FCR and the time of its arrival at the admission buffer.

5.5.1 Fork and Join Elements

The parallelization of queries is captured with fork and join elements. The *JobForkStation* element forks a job into a number of *tasks* and essentially models a client query being broken up into smaller tasks by the query planner. We assume cores to have identical processing speeds and jobs to have equal routing probabilities to each core. The number of tasks forked for job j at cycle $i \equiv i(j)$ is denoted by $f_{j,i}$ that is a positive integer. Similarly, the time demand placed by each task at the core at cycle i is indicated with $d_{j,i}$. After leaving the cores, all tasks are progressively accumulated by the *JobJoinStation* element until all $f_{j,i}$ tasks have arrived. At this point, they are merged back into job j which is then either moved to the next cycle $i(j)+1$ or departs the network if $i(j) = I(j)$. Thus, the time to complete cycle i equals the maximum

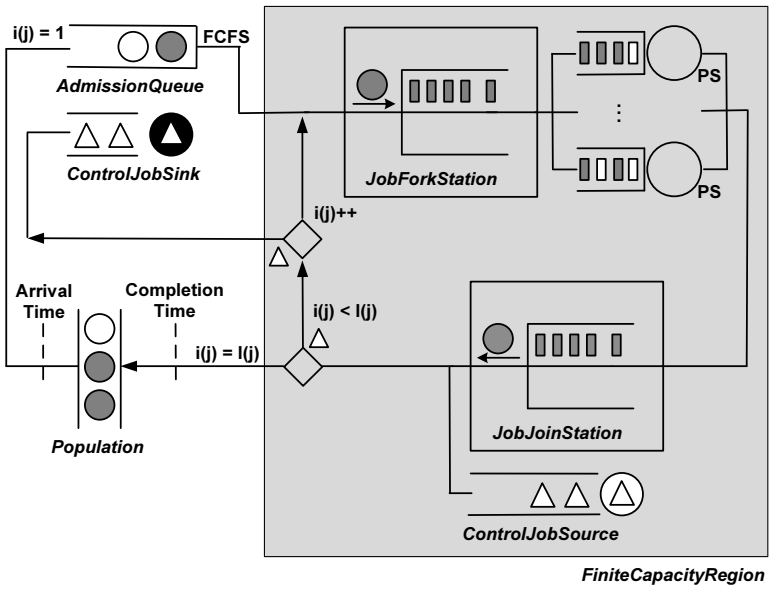


Figure 5.2: Simulation Model.

residence time of the $f_{j,i}$ forked tasks in the queues modelling the cores.

5.5.2 Finite Capacity Region

The FCR models a constraint on the quantity of available software thread resources in the database application. We implement this restriction with a global counter tracking how many forked jobs have entered the FCR region. Jobs positioned in the *AdmissionQueue* are granted FCR access if the nominal job fork size, i.e. the fork count at iteration $i = 1$, does not exceed the available FCR capacity. That is, let

$$n_{fcr}(t) = \sum_{j \in \text{ jobs in FCR}} f_{j,1} \tag{5.2}$$

be the number of jobs in execution inside the FCR at time t. Then the FCR imposes

$$n_{fcr}(t) \leq N_{fcr}, \quad \forall t$$

where N_{fcr} is the total FCR capacity, an input parameter of the model that limits the concurrency level inside the region. Tasks from different jobs concurrently executing on the cores may place different service demands and are therefore likely to complete execution at different times and with different orders with respect to the arrival order. Our model introduces control signals that are triggered as soon as a forked task arrives at the *JobJoinStation* element. The control signals decrement the FCR counter and are routed from the *ControlJobSource* to the *ControlJobSink* elements in the model as shown in Figure 5.2.

Table 5.1: Summary Reference System.

Hardware	Architecture	IBM X5, 4-Sockets
	Memory	1TB RAM
	CPU	32 Cores (64 Hyperthreads)
	Chipset	Intel Nehalem 7560, 8 cores at 2.27GHz
Software	Operating System	SLES 11
	In-Memory DB	HANA DB (GA) CL 351658 1.00
	Size Data Set Uncompressed	1.3TB
	Size Data Set Compressed	80GB

5.6 Simulation Model Validation

5.6.1 Reference System

For validation of our methodologies, we used a dataset from benchmark experiments performed on a four socket IBM X5 server with Intel Nehalem 7560 chipset and 1TB of RAM. This dataset was obtained and studied in [116]. The server features 32 physical cores, each clocked at 2.27Ghz with hyperthreading enabled, and so can leverage 64 hyperthreads. (In what follows, we use the terms core and hyperthread interchangeably.) The in-memory database used in the experiments is SAP HANA DB in the General Availability (GA) version 1.00. The high-level architecture of this system is illustrated in Figure 5.3. Once a database client submits a query request, it first enters the internal admission buffer of the *IndexServer*. This software component maintains the query execution control and is responsible for planning and management of query execution steps. Query planning is essential to database performance as complex queries may require multiple separate steps to complete (selects, joins, ...). The *IndexServer* decides this parallelism level in such a way as to utilize cores efficiently. A configurable maximum threading level is maintained in order to control the parallelism level. Depending on the job mix currently in execution and on task parallelization levels when the query plan is completed, queries issued to HANA that cannot be served by the available threads wait in the admission buffer. See Table 5.1 for a summary of the hardware and software environment.

5.6.2 Workload

The workload is based on the TPC-H [12] benchmark which emulates workloads of online analytical processing (OLAP) systems. This benchmark defines $K = 22$ query classes. For amplification of the analytical workload, additional columns were added to the database tables *lineitem* and *orders*. The database was populated with a 1.3TB dataset (80GB working set in

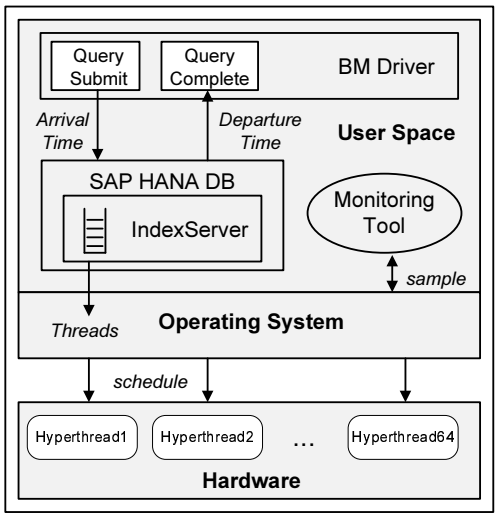


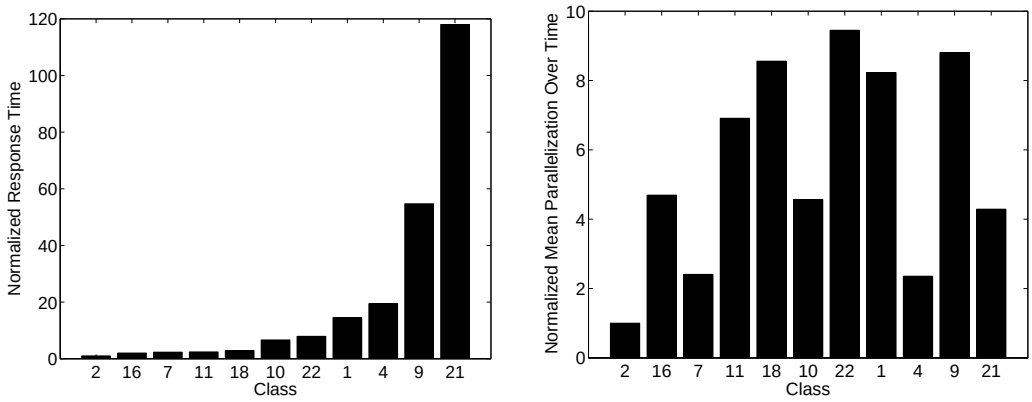
Figure 5.3: Reference System.

memory after HANA DB compression). Benchmark experiments were executed with different numbers of clients. In single client mode, denoted CON1 (concurrency level 1), the benchmark driver at any time has only a single outstanding query (i.e., job) in the system. In concurrent user mode, multiple client streams submit jobs so that job response times are affected due to contention. A client randomly chooses the class of the next job and upon its completion submits the subsequent job without delay. Two scenarios with 8 and 16 concurrent clients, respectively denoted CON8 and CON16, were available in the dataset.

All benchmarks are executed over a period of 1 hour during which timestamps were recorded for query arrival and departure events. The dataset reports thread status, i.e., affinity to a certain core, in sampling intervals of $0.2s$. This sampling rate may appear coarse grained for reporting core thread affinities, but is appropriate for the workload under study as it consists of queries with mean processing times many times larger than the sampling rate.

The workload contains a set of database queries with quite diverse characteristics. Due to the large number of classes and some long execution times, the measurements for some classes were limited to a few points. We determine the confidence of the measurements using 95% confidence intervals (CIs) and compute the CI widths as relative values to the mean. Figure 5.4 reports measurements for classes producing the reasonably low CI widths ranging in $[24\%; 130\%]$. Measurements are normalized by the values of class 2.

Figure 5.4(a) conveys that the range of query response times is large. A set of short running classes have small response times close to the base class. Classes $\{10, 22, 1, 4\}$ can be characterized as medium length runners and have response times up to 20 times larger than class 2. Finally we record the two significantly longest running classes $\{9, 21\}$. The diversity in workload classes is further highlighted by the measured parallelization levels. Figure 5.4(b) shows



(a) Response Times (b) Parallelization Levels

Figure 5.4: Single Client Measured Response Times and Parallelization Levels for Query Subset.

the mean parallelization levels over query processing phases as described in Section 5.6.3 and illustrates that parallelization levels induced by the query planner are unique for each class. Interestingly, there is no clearly visible relationship between query response time and level of parallelization.

5.6.3 Validation

For validation of the proposed simulation model, we simulate the CON8 and CON16 scenarios and compare simulation results to measurements. The model input parameters job *fork size* and *service time* are computed from core thread affinity traces included in the dataset. In particular we use data from isolation runs, i.e. with only a single query in execution, where over the length of the run the total number of active threads pertaining to the query process are recorded. Figure 5.5 illustrates the model parameterization methodology. The number of threads during query execution may change depending on parallelization levels in the query plan. We count each change in parallelization levels, i.e. number of active threads, as a separate processing phase of the query. In measurements we keep a record of the number of active threads per processing phase and the length of time the processing phase lasted. In simulation the number of active threads in a given processing phase and the time interval during which the phase was observed are used as parameters for fork sizes $f_{j,i}$ and service times $d_{j,i}$, respectively. The measured query processing phases map to the modeling concept of job iterations, denoted $i(j)$.

Furthermore, our reference system maintains 64 cores and consequently our model comprises 64 processor sharing queues: processor sharing is a well known idealization of CPU scheduling in computer systems [57]. Additional input parameters for the model are job think times, job population size (N), and FCR threading limit (N_{fcr}). For think times we use mea-

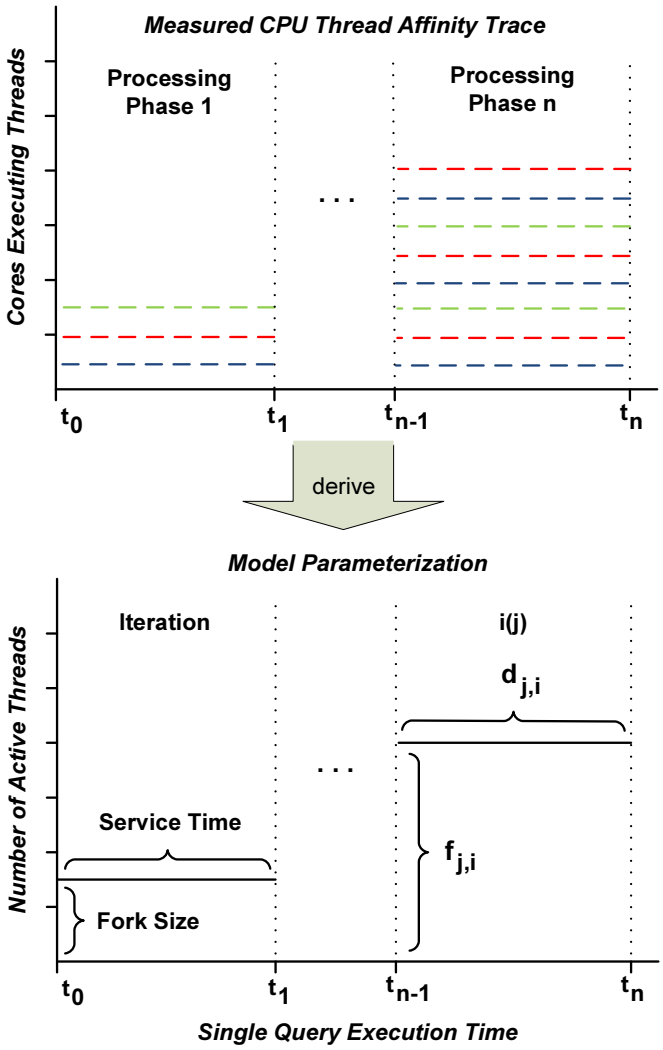
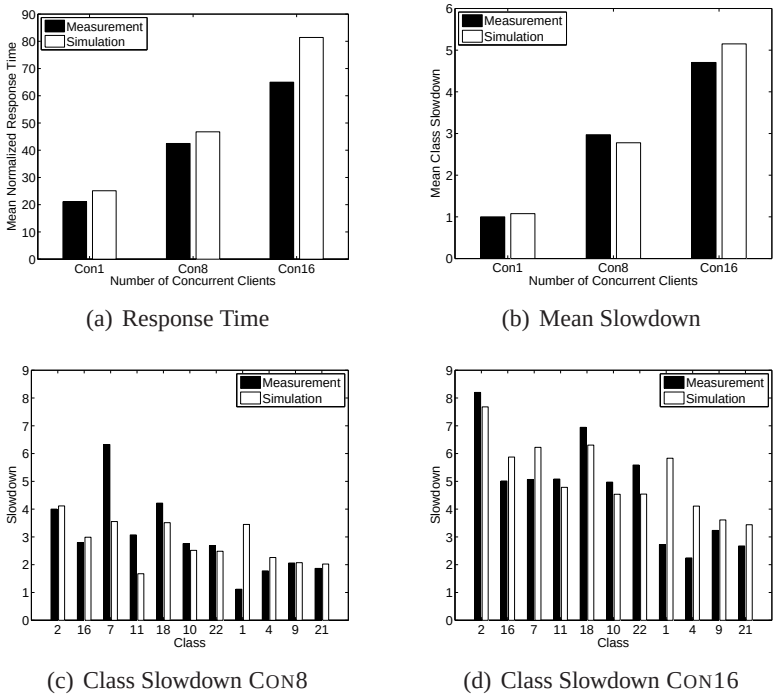


Figure 5.5: Model Parameterization Methodology.



(a) Response Time (b) Mean Slowdown

(c) Class Slowdown CON8 (d) Class Slowdown CON16

Figure 5.6: Mean Response Time and Slowdown in Measurement and Simulation.

sured traces from CON8 and CON16, respectively. Populations are estimated from measured throughput and response times using Little's law [130] ($N = 41$ for CON8 and $N = 60$ for CON16). The FCR parameter N_{fcr} was unknown, here the value producing similar mean job arrival queues in simulation compared to the CON8 and CON16 data was chosen ($N_{fcr} = 180$ for CON8 and $N_{fcr} = 275$ for CON16).

Our simulation experiments show that the model captures measured contention delays reasonably well. Figure 5.6(a) reveals that simulation tends to overestimate mean response times. Define the slowdown S_k of class-k queries as the ratio between that class's response time and its demand D_k, estimated as the response time in the CON1 experiment. As shown in Figure 5.6(b), the mean class slowdown is matched very accurately: we record simulation errors for CON8 and CON16 of 0.06 and 0.10, respectively. To better understand the source of the per-class errors, Figures 5.6(c)-(d) compare per-class slowdowns from measurement and simulation. The model accurately captures variances in slowdown values and closely matches most measurements. Class 7 appears to be difficult to model in CON8. Simulation data of Class 1 overestimates measurements in both CON8 and CON16 scenarios. Considering the complexity of the model topology and the total of 22 considered classes, low deviations from measurement appear of sufficient quality for the purpose of defining a validation environment for the admission control policies.

Table 5.2: Fork Sizes ($f_{j,i}$) and per Core Service Times ($d_{j,i}$) of Synthetic Workload Classes.

	Var-Service		*Var-Para*		*Inv_Increase*		*Lin_Increase*	
Class	$f_{j,i}$	$d_{j,i}$	$f_{j,i}$	$d_{j,i}$	$f_{j,i}$	$d_{j,i}$	$f_{j,i}$	$d_{j,i}$
SC1	10	2	5	10	5	10	5	2
SC2	10	5	10	10	10	5	10	5
SC3	10	10	15	10	15	2	15	10

5.7 Comparative Evaluation of Resource Management Methodologies

We now use our simulation model for a comparative evaluation of job admission policies. The first part of this section applies simplified synthetic workloads before progressing in the second part to more realistic workload mixes based on measurements from the reference system, described in Section 5.6.2. In simulation the job scheduling is performed in the *AdmissionQueue* element depicted in Figure 5.2. We quantify the performance using throughput, response time and weighted slowdown metrics. Weighted slowdown explains how much the slowdown of a particular class affects the workload mix and is defined as

$$W(S_k) = S_k \frac{X_k}{X}, \tag{5.3}$$

where X_k and X describe the class k and total throughputs, respectively. Thus low $W(S_k)$ values are preferable. We compare the admission policies $[P_{r,c}]$, $[U_r]$, $[-A_r^c]$, and $[-Q_r]$ to the traditional policies FCFS and SEPT. In related work, shortest job prioritization was applied successfully as admission control policy [84].

5.7.1 Synthetic Workloads

The study of synthetic workloads facilitates the creation of simplified example cases and aids in the interpretation of admission policy prioritizations. In particular, we define the four workload types specified in Table 5.2 consisting of three classes each, denoted *SC1*, *SC2*, and *SC3*. The synthetic classes differ in terms of the parameters fork size $f_{j,i}$ and service time $d_{j,i}$:

- $f_{j,i}$ quantifies the number of tasks job j is forked into at his ith circle to the servers
- $d_{j,i}$ specifies the service time of a job j task at his ith circle to the servers

The first workload type, denoted *Var-Service*, is configured with static fork size and altering service time parameter, while *Var-Para* varies fork sizes and maintains the service times stable for all classes. Both parameter dimensions are configured to be dynamic in the workloads denoted *Inv_Increase* and *Lin_Increase*. The maximum number of cycles each job performs in

the FCR is set to $I(j) = 1$. Think times are generated from an exponential distribution with a mean of five.

For each workload we obtain three distinct mixes, denoted $[\text{Mix-I}, \text{Mix-II}, \text{Mix-III}]$, by applying the probability vectors $[0.6, 0.2, 0.2]$, $[0.2, 0.6, 0.2]$, $[0.2, 0.2, 0.6]$ specifying how many jobs of classes $[SC1, SC2, SC3]$ are in the mix. Intuitively, the scalar in the identifier of the mix encodes which synthetic class holds the largest number of jobs in the workload, e.g., Mix-I has a majority of class SC1 jobs. Our simulations use a population of $N = 30$ jobs, $n = 64$ servers, and the FCR constraint limit $N_{fcr} = 100$. The $\boldsymbol{\eta}$ tables are built in FCFS simulations.

5.7.1.1 Simulation Results Workload Var-Service

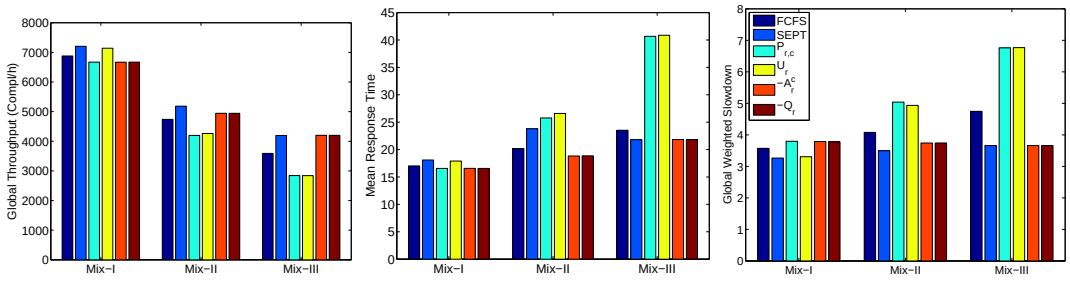
This workload features constant fork size and varying service time parameters. The performance results and admission policy statistics for Var-Service are illustrated in Figures 5.7(a)-(c) and Table 5.3, respectively.

$[P_{r,c}]$. We find this policy prioritizing the classes with largest service requirements resulting in low throughputs especially in the case of Mix-III. The mean response times and weighted slowdowns in Mix-II and Mix-III are considerably larger compared to the conventional policies SEPT and FCFS. The policy produces competitive response times in Mix-I.

$[U_r]$. Prioritizing classes with high utilizations delivers good performance only in Mix-I, where the majority of jobs are of the class with the smallest service time, i.e. SC1. However, U_r does not admit jobs in the same order as SEPT in Mix-I and assigns higher priority to SC3 over SC2. In Mix-I and Mix-II we observe equal prioritizations for $[U_r]$ and $[P_{r,c}]$ resulting in similar performance.

$[-A_r^c]$, $[-Q_r]$. The statistics presented in Table 5.3 convey that arrival instant queue lengths and time averaged queue lengths are similar. The admission policies leverage unique prioritization patterns for each of the studied workload mixes. Interestingly, the throughput performance is competitive to SEPT in Mix-I and Mix-II in spite of different prioritizations. We record the best response time performance using this approach. Weighted slowdowns are considerably smaller compared to the utilization based policies in Mix-II and Mix-III.

Interpretation. The two approaches leveraging utilization consistently prioritize the jobs with highest service requirements. Not surprisingly this leads to low mean throughputs and large response times: short jobs are trapped behind the longer ones. The queue based policies favor jobs that over time build up the smallest backlogs and therefore choose the classes imposing the smallest additional load to the running mix. This strategy leads to different prioritizations than SEPT for Mix-I and Mix-II resulting in competitive throughputs and favorable response times.



(a) Throughput (b) Response Time (c) Weighted Slowdown

Figure 5.7: Synthetic Workload Var-Service Global Throughput, Mean Response Time, and Weighted Slowdown in Simulation. The Legend Shown in Figure (c) is Identical for Figures (a) and (b).

Table 5.3: Synthetic Workload Var-Service Table-Row Sum ($\sum_{c=1}^{K}$), Mean Class Utilization (U_r), Mean Task Arrival Instant Queue Length (A_r^c), and Task Time Averaged Queue Length (Q_r) as Used to Compute Admission Policy High (3), Medium (2), and Low (1) Prioritizations (Prio). Priorities High, Medium, Low are respectively highlighted in Red, Amber, and Green.

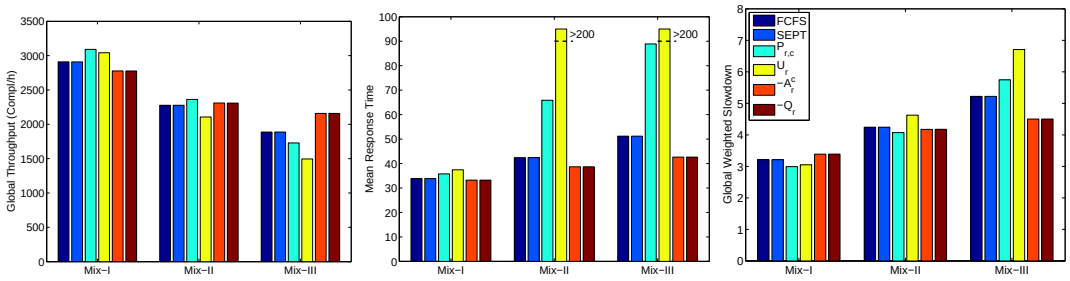
		Admission Policy							
		$P_{r,c}$		U_r		$-A_r^c$		$-Q_r$	
	Class	$\sum_{c=1}^{K}$	*Prio*	U_r	*Prio*	A_r^c	*Prio*	Q_r	*Prio*
Mix-I	SC1	-0.19	1	0.45	3	46.5	1	43.4	1
	SC2	0.02	2	0.23	1	21.9	3	22.2	3
	SC3	0.04	3	0.29	2	27.2	2	27.9	2
Mix-II	SC1	-0.08	1	0.12	1	14.0	3	12.0	3
	SC2	0.33	3	0.59	3	59.1	1	58.1	1
	SC3	0.24	2	0.26	2	24.7	2	25.2	2
Mix-III	SC1	-0.09	1	0.10	1	12.4	3	10.1	3
	SC2	0.16	2	0.17	2	17.8	2	17.1	2
	SC3	1.05	3	0.69	3	67.7	1	68.3	1

Key points for Var-Service:

- $[-A_r^c]$ and $[-Q_r]$ quantities and prioritizations are very similar
- $[P_{r,c}]$ and $[U_r]$ show worst overall performance
- SEPT has high throughputs but some large response time values
- $[-A_r^c]$ and $[-Q_r]$ competitive for throughputs and best response time results

5.7.1.2 Simulation Results Workload Var-Para

In workload Var-Para the fork size parameters are configured to uniformly increase from class SC1 to SC3, while the service times remain constant. The performance results of our simulation



(a) Throughput (b) Response Time (c) Weighted Slowdown

Figure 5.8: Synthetic Workload Var-Para Global Throughput, Mean Response Time, and Weighted Slowdown in Simulation. The Legend Shown in Figure (c) is Identical for Figures (a) and (b).

Table 5.4: Synthetic Workload Var-Para Table-Row Sum ($\sum_{c=1}^{K}$), Mean Class Utilization (U_r), Mean Task Arrival Instant Queue Length (A_r^c), and Task Time Averaged Queue Length (Q_r) as Used to Compute Admission Policy High (3), Medium (2), and Low (1) Prioritizations (Prio). Priorities High, Medium, Low are respectively highlighted in Red, Amber, and Green.

		Admission Policy							
		$P_{r,c}$		U_r		$-A_r^c$		$-Q_r$	
	Class	$\sum_{c=1}^{K}$	*Prio*	U_r	*Prio*	A_r^c	P	Q_r	*Prio*
Mix-I	SC1	0.0005	2	0.40	3	41.1	1	40.8	1
	SC2	0.03	3	0.24	1	23.8	3	23.1	3
	SC3	-0.06	1	0.33	2	33.3	2	31.5	2
Mix-II	SC1	-0.01	1	0.11	1	11.2	3	11.1	3
	SC2	0.22	3	0.59	3	59.2	1	57.5	1
	SC3	0.02	2	0.28	2	27.9	2	26.5	2
Mix-III	SC1	-0.03	1	0.09	1	9.45	3	9.41	3
	SC2	0.08	2	0.17	2	16.8	2	16.4	2
	SC3	0.26	3	0.72	3	71.4	1	68.1	1

experiments using this workload are depicted in Figures 5.8(a)-(c). Table 5.4 shows statistics of the evaluated admission control policies.

$[P_{r,c}]$. The no-interference baseline based approach delivers the maximal throughputs in Mix-I and Mix-II, where it assigns highest priority to SC2: the class with medium fork size. While this prioritization is successful in terms of throughput and weighted slowdown, it produces large response times in Mix-II and Mix-III where classes with largest fork sizes are admitted first. The poor response time performance may result from classes with high fork sizes claiming large fractions of the FCR limit causing delays for the remaining classes.

$[U_r]$. This policy overall delivers the poorest performance for workload Var-Para. In Mix-II and Mix-III this choice of admission results in low throughputs, high response times, and large weighted slowdowns. The explanation for these results may be the prioritization of classes

with highest fork counts. While all classes have identical per core service requirements, this admission order saturates the FCR limit at the expense of classes with smaller fork sizes.

$[-A_r^c], [-Q_r]$. The queue lengths based techniques appear to be a good choice for workloads with variable fork sizes. Our simulation results reveal high throughputs and low response times especially in Mix-II and Mix-III where the majority of jobs have large fork sizes. Since all jobs have identical per core service requirements these policies make best use of the FCR resource by prioritizing the job classes building up the smallest backlogs. Interestingly, job classes with smallest fork sizes do not always get prioritized FCR entry. For example, in Mix-I and Mix-II SC3 does not receive the lowest priority. In terms of weighted slowdown this policy is competitive in Mix-I and proves to be the best choice for Mix-II and Mix-III.

Interpretation. Assigning highest priorities to the classes with largest parallelization levels is not an optimal strategy. The query-pair and utilization based policies follow this approach for Mix-II and Mix-III resulting in poor response time performance. We interpret this finding with the thread pool limit constraint causing delays for requests waiting for other highly parallel jobs to complete. Instead, the queue based policies privilege the least parallel class SC1 in Mix-I and Mix-II leading to better throughputs and response times than any other policy.

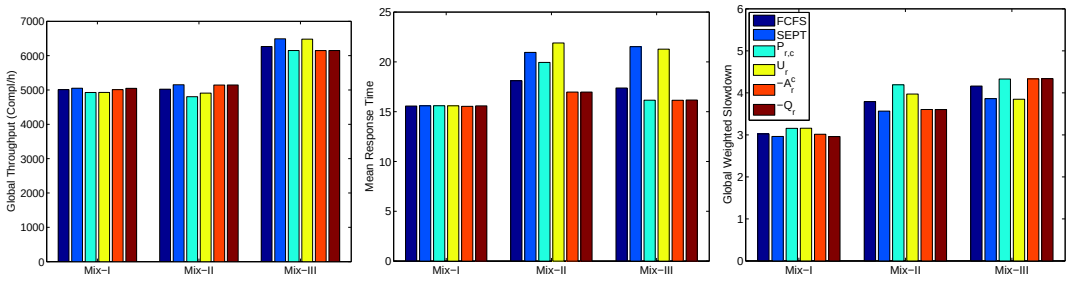
Key points for Var-Para:

- $[-A_r^c]$ and $[-Q_r]$ quantities and prioritizations are very similar
- $[U_r]$ delivers very large response time results
- $[P_{r,c}]$ is not competitive with $[-A_r^c]$ and $[-Q_r]$ in Mix-II and Mix-III
- SEPT behaves identical to FCFS in this setting performing worse than $[-A_r^c]$ and $[-Q_r]$ especially in Mix-III with mostly highly parallelized jobs
- $[-A_r^c]$ and $[-Q_r]$ produce best overall results for throughput, response time, and slowdown

5.7.1.3 Simulation Results Workload Inv_Increase

In workload Inv_Increase the combination of fork size and service time parameters is configured uniquely and inverse for each class. For example, SC1 has the smallest fork size and largest service requirement, while SC3 has the largest fork size and smallest service requirement. Illustrations of simulation results and statistics of the admission policies can be found in Figures 5.9(a)-(c) and Table 5.5, respectively.

$[P_{r,c}]$. The lowest priority is consistently assigned to class SC3: The one with largest fork size and smallest service time. The strategy of prioritizing classes with largest per task service times produces competitive throughputs and favorable response times compared to SEPT admission across all workload mixes.



(a) Throughput (b) Response Time (c) Weighted Slowdown

Figure 5.9: Synthetic Workload Inv_Increase Global Throughput, Mean Response Time, and Weighted Slowdown in Simulation. The Legend Shown in Figure (c) is Identical for Figures (a) and (b).

Table 5.5: Synthetic Workload Inv_Increase Table-Row Sum ($\sum_{c=1}^{K}$), Mean Class Utilization (U_r), Mean Task Arrival Instant Queue Length (A_r^c), and Task Time Averaged Queue Length (Q_r) as Used to Compute Admission Policy High (3), Medium (2), and Low (1) Prioritizations (Prio). Priorities High, Medium, Low are respectively highlighted in Red, Amber, and Green.

		Admission Policy							
		$P_{r,c}$		U_r		$-A_r^c$		$-Q_r$	
	Class	$\sum_{c=1}^{K}$	*Prio*	U_r	*Prio*	A_r^c	*Prio*	Q_r	*Prio*
Mix-I	SC1	0.27	3	0.48	3	46.4	1	47.2	1
	SC2	0.13	2	0.24	2	22.4	3	22.7	2
	SC3	-0.19	1	0.22	1	24.6	2	21.0	3
Mix-II	SC1	0.11	2	0.15	1	14.6	3	15.0	3
	SC2	0.55	3	0.63	3	61.0	1	61.5	1
	SC3	-0.28	1	0.19	2	21.6	2	18.0	2
Mix-III	SC1	-0.03	2	0.15	1	15.1	3	15.4	3
	SC2	0.11	3	0.22	2	20.8	2	21.5	2
	SC3	-0.34	1	0.60	3	61.0	1	56.3	1

$[U_r]$. Our simulation results for throughput and response times reveal similar performance characteristics to SEPT admission for this policy. This finding is particularly interesting, since the prioritization patterns are remarkably different to SEPT in case of Mix-I and Mix-II. However, choosing U_r or SEPT as admission policy delivers the worst response time performance for workload Inv_Increase.

$[-A_r^c], [-Q_r]$. This policy once more delivers the best performance for mean response time. In addition, it is competitive to SEPT in global throughputs and weighted slowdown. In Mix-II and Mix-III the choice of admission is similar to $[P_{r,c}]$ in that classes with small fork sizes and large service times are prioritized.

Interpretation. The SEPT policy prioritizes the short and highly parallel jobs leading to high saturation levels of the thread pool limit. We find this to be the worst strategy resulting in

poor response time performance. On the contrary, the query pair policy prioritizes the classes with little parallelization and high service requirements: we interpret these to be the work-intensive jobs. Prioritizing the work-intensive mixes leads to competitive throughputs and with the exception of Mix-II to the joint lowest response times. Similarly to the query pair based policy the queue length approaches prioritize jobs with small parallelization levels.

Key points for Inv_Increase:

- Minor difference in $[-A_r^c]$ and $[-Q_r]$ prioritizations
- SEPT and $[U_r]$ with worst response time performance
- $[P_{r,c}]$ performance similar to $[-A_r^c]$ and $[-Q_r]$
- $[-A_r^c]$ and $[-Q_r]$ overall produce best and most stable results

5.7.1.4 Simulation Results Workload Lin_Increase

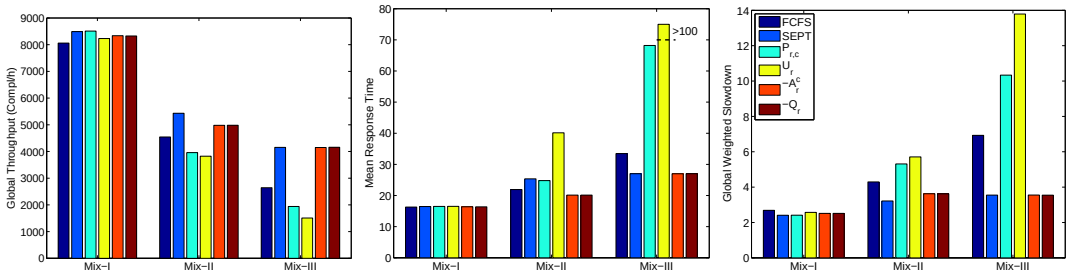
The fork size and service time parameters of workload Lin_Increase are configured as linearly increasing along both dimensions. Table 5.6 presents the statistics of the admission policies and Figures 5.10(a)-(c) provide illustrations of the performance metrics from simulation.

$[P_{r,c}]$. The throughput performance of this admission policy is low in Mix-II and Mix-III, while response times are competitive with the other policies except for the case of Mix-III. We find lowest table-row sums for class SC3 which is configured with the largest fork size and service time.

$[U_r]$. Building admission policies solely based on job utilization again proves to be a bad choice. Our results show lowest throughputs paired with highest response times and large slowdowns for this policy.

$[-A_r^c], [-Q_r]$. The performance results of the queue based policies are excellent for throughputs and consistent in beating SEPT for lower response times. The prioritization favors the classes with smaller fork sizes facilitating small delays for available FCR capacities. Weighted slowdown results show a clear improvement compared to the utilization based policies.

Interpretation. Due to the workload characteristics the prioritizations of $[-A_r^c]$, $[-Q_r]$, and SEPT are quite similar: The classes building up the least amount of backlogs also have smallest service times. Results for the utilization based policy are significantly worse, since it assigns high priorities to the classes with largest parallelization levels and service requirements. This strategy most likely results in large fractions of the limited threads being blocked for long periods of time.



(a) Throughput (b) Response Time (c) Weighted Slowdown

Figure 5.10: Synthetic Workload Lin_Increase Global Throughput, Mean Response Time, and Weighted Slowdown in Simulation. The Legend Shown in Figure (c) is Identical for Figures (a) and (b).

Table 5.6: Synthetic Workload Lin_Increase Table-Row Sum ($\sum_{c=1}^{K}$), Mean Class Utilization (U_r), Mean Task Arrival Instant Queue Length (A_r^c), and Task Time Averaged Queue Length (Q_r) as Used to Compute Admission Policy High (3), Medium (2), and Low (1) Prioritizations (Prio). Priorities High, Medium, Low are respectively highlighted in Red, Amber, and Green.

		Admission Policy							
		$P_{r,c}$		U_r		$-A_r^c$		$-Q_r$	
	Class	$\sum_{c=1}^{K}$	*Prio*	U_r	*Prio*	A_r^c	*Prio*	Q_r	*Prio*
Mix-I	SC1	0.07	3	0.27	2	26.6	2	25.6	2
	SC2	-0.03	2	0.24	1	23.2	3	23.0	3
	SC3	-0.06	1	0.41	3	38.5	1	39.5	1
Mix-II	SC1	-0.002	2	0.06	1	6.82	3	6.44	3
	SC2	0.17	3	0.56	3	57.2	1	55.1	1
	SC3	-0.02	1	0.35	2	33.4	2	33.8	2
Mix-III	SC1	-0.01	2	0.04	1	4.13	3	3.82	3
	SC2	0.18	3	0.13	2	13.7	2	12.8	2
	SC3	-0.10	1	0.80	3	79.0	1	77.7	1

Key points for Lin_Increase:

- $[-A_r^c]$ and $[-Q_r]$ quantities and prioritizations are very similar
- $[U_r]$ delivers a very large response time result in Mix-III
- $[P_{r,c}]$ is not competitive with $[-A_r^c]$ and $[-Q_r]$ in Mix-II and Mix-III
- SEPT, $[-A_r^c]$, and $[-Q_r]$ perform equally well with the exception of Mix-II where the response times from the queue based policies are lower

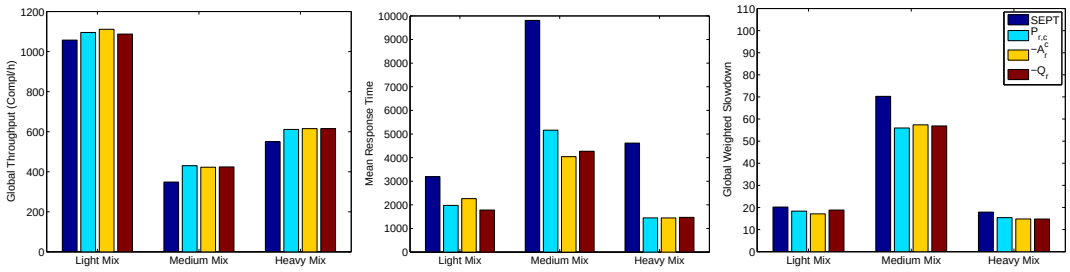
5.7.1.5 Summary Synthetic Workloads

Our comparative evaluation of admission control policies conveys that the proposed table based approaches can deliver improved performance compared to FCFS and SEPT scheduling in simulations. While building the admission table from class utilization $[U_r]$ produced some less favorable results in our study, the no-interference baseline table from query pair arrival instant queue lengths and utilization ratios showed its potential especially with workload Inv_Increase and workload Lin_Increase. These workloads are the most important in terms of our application scenario as they are configured to vary both service time and fork size parameters. Our simulations further convey that performance results may be improved using the queue lengths techniques $[-A_r^c]$ and $[-Q_r]$. In particular, we record the most stable results and lowest response times using this approach.

This in an interesting result of our study, since scheduling policies typically do not consider thread pool limits and job parallelization, but rely on prioritizations based on request sizes and service demands. Conversely, load balancing policies consider the queue lengths at the destination servers. However, we consider a reference system with external admission control and without control over the destination environment. Our simulation results indicate that the presented queue length policies are a viable approach for external admission control in presence of finite capacity regions and high variance in job parallelization levels. In addition, we find that building $\boldsymbol{\eta}$ tables from arrival instant queue lengths and time averaged queue lengths produces very similar results. Using the time averaged queue lengths for admission policy prioritization has the advantage of less overheads, since rather than maintaining a table this metric facilitates storing of a single backlog value for each workload class.

5.7.2 Customized TPC-H Based Workloads

In this section we progress to more realistic workload definitions and obtain fork size and service time parameters from the reference workload, introduced in Section 5.6.2. Our evaluation focuses on the best admission policies identified in the previous study of synthetic workloads: SEPT, $[P_{r,c}]$, $[-A_r^c]$, and $[-Q_r]$. For ease of interpretation of the experiment we consider a subset of six classes. In particular we use classes $\{16, 18\}$, $\{1, 4\}$, and $\{9, 21\}$ to represent short, medium, and long running query types, respectively. We regard three workload mixes with a population of $N = 200$ jobs denoted *Light*, *Medium*, and *Heavy Mix*. The mixes are defined by the probability vectors $[0.6, 0.2, 0.2]$, $[0.2, 0.6, 0.2]$, $[0.2, 0.2, 0.6]$ specifying how many jobs of $[short, medium, long]$ running types are in the respective mix. We specifically chose workloads with high variance in job service demands. For example, Figure 5.4(a) reveals that class 21 response times in CON1 are roughly 8 times larger than class 1 and 120 times larger than class 2 response times. SEPT admission prioritizes job classes with smaller



(a) Throughput (b) Response Time (c) Weighted Slowdown

Figure 5.11: Customized TPC-H Type Workload Global Throughput, Mean Response Time, and Weighted Slowdown in Simulation. The Legend Shown in Figure (c) is Identical for Figures (a) and (b).

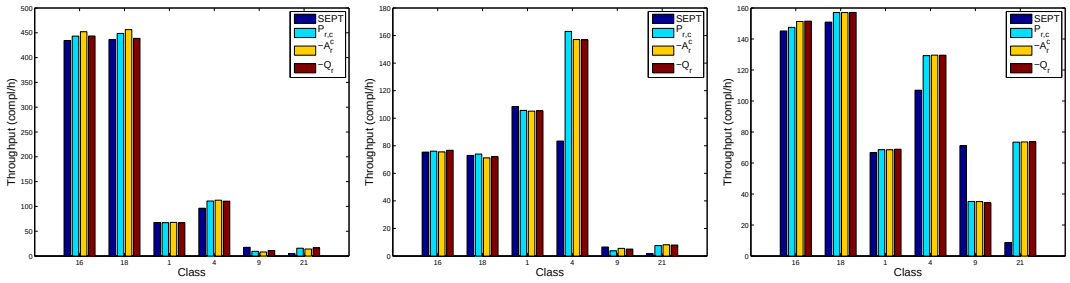
Table 5.7: TPC-H Based Workload Normalized CON1 *Response Time (R_{norm}), Table-Row Sum ($\sum_{c=1}^{K}$), Mean Task Arrival Instant Queue Length (A_r^c), and Task Time Averaged Queue Length (Q_r) as Used to Compute Admission Policy Highest (6) to Lowest (1) Prioritizations (Prio).*

	Admission Policy							
	SEPT		$P_{r,c}$		$-A_r^c$		$-Q_r$	
Class	R_{norm}	*Prio*	$\sum_{c=1}^{K}$	*Prio*	A_r^c	*Prio*	Q_r	*Prio*
16	1.0	6	1.4	6	5.0	5	1.6	6
18	1.4	5	0.2	4	12.1	4	6.8	4
1	7.2	4	-1.5	3	14.3	3	10.1	3
4	9.7	3	1.3	5	4.3	6	3.1	5
9	27.4	2	-63.9	1	78.5	1	57.8	1
21	59.0	1	-29.4	2	38.5	2	43.9	2

CON1 response times. The $\boldsymbol{\eta}$ matrix tables are built from CON16 measurements.

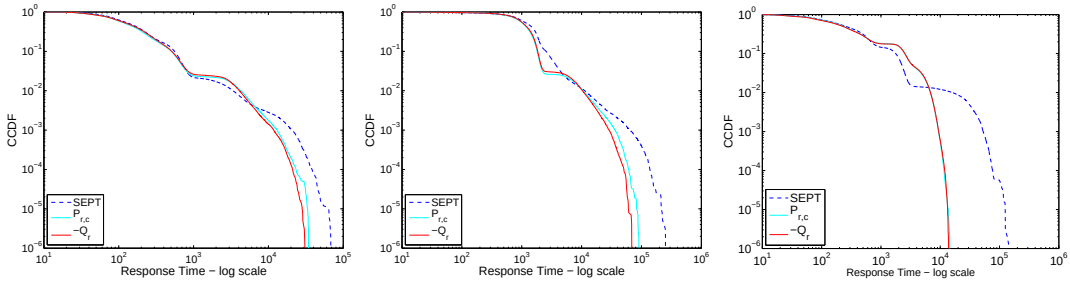
Throughput Result. The global throughput results presented in Figure 5.11(a) show that our proposed admission policies increase global throughputs compared to SEPT. This finding is particularly interesting in the Light Mix where largest throughputs would naturally be expected with highest priorities for short jobs. The prioritization statistics shown in Table 5.7 reveal that $[P_{r,c}]$, $[-A_r^c]$, and $[-Q_r]$ assign prioritized access to class 4 over the shorter running classes 18 and 1. In addition and contrary to SEPT, class 21 is granted access ahead of class 9. To better understand the performance effects of our admission policies we study the per class throughputs illustrated in Figures 5.12(a)-(c). Prioritizing classes 4 and 21 appears to be most effective in the Medium and Heavy Mixes, respectively depicted in Figures 5.12(b) and (c). With the exception of class 9, all remaining classes benefit from this choice.

Response Time Result. Simulation results indicate that mean response times can be significantly decreased using our proposed admission policies. Compared to SEPT we record lower mean response times across all workload mixes. To verify that our policies do not lead to



(a) Light Mix (b) Medium Mix (c) Heavy Mix

Figure 5.12: Throughputs in Simulations with TPC-H Type Custom Workloads.



(a) Light Mix (b) Medium Mix (c) Heavy Mix

Figure 5.13: Complementary Cumulative Distribution Function (CCDF) of Response Times in Simulation with Shortest-Expected-Processing-Time-First (SEPT), No-Interference Baseline ($P_{r,c}$), and Time Averaged Queue ($-Q_r$) Policies.

the starvation of low priority job classes we studied the tail distributions of response times in simulation. The results of the complementary cumulative distribution function, reported in Figures 5.13(a)-(c), reveal that probabilities for large response times are significantly lower than for SEPT scheduling in all workload mixes.

Weighted Slowdown Result. Figure 5.11(c) shows the weighted slowdowns are similar in the Light and Heavy Mixes with a slight advantage for $[P_{r,c}]$, $[-A_r^c]$, and $[-Q_r]$. However, we observe a substantial difference in the Medium Mix, where our policies produce lower weighted slowdowns compared to SEPT.

Summarizing. Simulation results with TPC-H type workloads indicate that system performance can be increased with the proposed resource management methodologies. Specifically we compare the $[P_{r,c}]$, $[-A_r^c]$, and $[-Q_r]$ policies to SEPT and find that total throughputs can be increased, while mean response times and weighted slowdowns are reduced.

Key points for TPC-H Based Workloads:

- $[P_{r,c}]$, $[-A_r^c]$ and $[-Q_r]$ increase global throughput compared to SEPT
- SEPT admission produces largest response times particularly in the Medium Mix

- Results of the queue based policies are slightly better than $[P_{r,c}]$
- $[-A_r^c]$ and $[-Q_r]$ record similar results

5.8 Summary and Conclusions

Conclusions. In this chapter we have proposed novel admission control policies for in-memory database systems. In particular, we make four contributions. First we develop admission policies based on queue lengths, utilization, and a query pair based no-interference baseline that aim to prioritize query mixes making efficient use of the system resources. The development of high performance policies is complicated by the complex interplay of database queries competing for processing resources and a limited number of software threads. In addition, typical workloads of such large memory systems entail high variance in service requirements and parallelization levels. Our policies ignore I/O subsystems and therefore are highly applicable to in-memory databases. Secondly, we present a trace-driven simulation model for evaluation of admission control policies. The simulation model features fork/join elements and captures limited thread pools with a FCR. The model is validated against system measurements.

Our third contribution is a comparative simulation study of the proposed admission control policies using a set of simplified synthetic workloads. Finally, we validate our techniques with trace-driven simulations and more realistic TPC-H type workloads build from measurements of the in-memory system SAP HANA. We find that our policies have the potential to significantly increase performance compared to the established policies FCFS and SEPT.

In particular, the policies considering job queue lengths deliver the best and most stable results in terms of response times and throughput. This is an important result of our study as external scheduling policies typically rely on service requirements for derivation of job prioritizations. This implies that queue lengths might be a better metric in systems with high variance in service demands, varying parallelization levels, and constraints on thread pool limits. Interestingly, the policy derived from the time averaged queue length delivers equal performance to the one using the arrival instant queue. This is a positive property of our methodology, since using the time averaged queue lengths entails less overhead.

The size based SEPT admission produces high throughputs particularly for workloads with small variance in parallelization levels. However, we record less favorable performance in cases where short jobs claim large fractions of the thread pool limit. The query pair based approach performs well with the TPC-H type workload, but has less favorable results in some simplified cases. This might be due to the use of the utilization metric in the no-interference baseline ratio. System utilization proves to be the worst choice for derivation of admission policies in our study.

Limitations. The proposed policies are limited to scenarios where system measurements of averaged thread queue lengths pertaining to a query class are available. Such measurements can be logged and made available by the operating system or the database application. The thread affinity monitoring used in our study employs the rather coarse grained sample interval of $0.2s$. Smaller sample intervals may deliver more realistic results, but can impose measurement overheads. In addition, our policies rely on the mapping of each arriving request to a class. In practice OLTP queries usually remain stable, while OLAP queries may be refined more often resulting in new class mappings. Related work offers clustering techniques for the assignment of database queries to classes [39]. Finally, our policies rely on a table generated during a training phase.

Usability. The proposed admission control policies can be used as a dispatcher external to the database. This approach has the advantage of treating the database as a black box. Therefore it is independent of implementation changes to the database internals.

6 Concluding Remarks

6.1 Conclusions

This thesis has presented a performance evaluation of data access operations in shared enterprise systems. It is structured into two parts addressing virtualized servers with disk storage and in-memory databases maintaining data sets in RAM. The first part of the thesis proposed predictive models of the performance of disk requests in virtualized environments. Our methodology uses measurements obtained within a VM running in isolation and effectively treats the VMM as a black box. Based on the measurements of the isolation case we then estimate the disk request performance of multiple VMs running in consolidation.

In particular, we presented models for homogeneous and heterogeneous workload consolidations. The homogeneous approach applies an open queueing model and trace-driven simulation. In the absence of detailed VMM level measurements, it is enhanced with a heuristic for model calibration accounting for inaccuracies in the model parameterization.

The heterogeneous approach is more powerful as it allows the prediction of workload consolidations of different types. In addition, we employ a decomposition technique to model read and write requests separately. The presented Decomposition models use a set of simple analytical estimators for prediction of averages of disk request latencies, throughputs, and read/write mixes. Furthermore, the model is complemented with a closed queueing model and trace-driven simulation for prediction of disk request latency distributions. All homogeneous and heterogeneous models are validated against system measurements.

Our models may assist administrators of virtualized data centres in finding placement configurations for VMs submitting disk I/O intensive workloads. In particular, the methodology requires the collection of system traces collected inside the VMs when running in an isolated environment. The traces are added to a trace repository and can subsequently be used for prediction of the disk I/O performance of VMs when they are consolidated on a single virtualized server.

In the second part of the thesis our performance study focused on systems storing data sets in system memory. We proposed and evaluated admission control policies for in-memory databases. Our policies rely on tables built from detailed measurements of system CPU traces thereby treating the database as a black box. In particular, we employ measurements of averaged job queue lengths and mean CPU utilization to derive four admission policies. We find that the queue length based policies deliver the most stable and best results in trace-driven simulations when compared to the established policies FCFS and SEPT.

For evaluation of our policies we used a closed queueing model with fork/join elements

and a FCR modeling limited thread pools. The simulation tool is validated against system measurements obtained on an enterprise server hosting the SAP HANA in-memory database. The evaluation of the admission policies comprised a comparative study based on simplified synthetic workloads and more realistic system traces recorded using a TPC-H type workload.

We see the proposed policies used as part of an admission controller deployed external to the database application. Database queries submitted from the clients may enter the admission controller where the admission order to the database application is altered according to the prioritizations of the admission policy. The admission controller aims to prioritize mixes of queries delivering good performance.

6.2 Directions for Further Research

A number of further research directions resulting from the presented work may be considered. We suggest the following extensions to the methodologies in Chapters 3 and 4:

- *Validation of the proposed methodologies on other virtualization platforms with similar system characteristics*. In particular, the fully virtualized environment *Parallels Cloud Server* [8] may be an interesting reference system for further validation of our methodology. In contrast to VMware ESX and Parallels, the hypervisor shipped with *VirtualBox* [17] is not installed on bare-metal, but requires an available operating system on the server. This kind of configuration is commonly referred to as 'hosted' hypervisor and may also be an interesting reference system for further validation of our proposed models.

- *Application to different storage technologies with block device interfaces*. The performance of data access operations can be significantly improved with modern storage devices omitting the mechanical movements inherent in traditional magnetic disks. *Solid State Drives (SSDs)* typically employ integrated circuits, i.e. NAND flash memory, and therefore do not require any movable mechanical parts. Recently *Storage Class Memory (SCM)* [89] has been proposed as an enterprise storage solution. This flash based technology can offer a block device interface and, e.g., promises performance improvements by omitting the traditional storage protocols, controllers, and interconnects [16].

- *Investigation of environments where storage is federated across multiple VMMs, i.e. physical hosts*. Related work has investigated how the consolidation of workloads submitted from multiple virtualized servers affects the performance of disk I/O requests [100]. It would be interesting to investigate how our models may be applied to environments where a number of VMs are hosted on different servers and share a storage device.

- *Capture of more realistic disc I/O workloads and consideration of production traces in addition to synthetic and emulated application workloads*. A possible extension into this

direction could be the consideration of real-world application traces published in third party repositories [15,27]. More realistic would be the validation with customer production traces. However, customer data sets are difficult to obtain in practice.

- *Investigation of how the predicted disk request performance at the VM operating system level correlates to end-to-end application performance*. This extension would require first the move from I/O benchmarks to a test application, e.g., *DVD Store* [2]. Using code profiling and system level tools the I/O intensive parts of the application may be identified. With statistical analysis, e.g. regression techniques, this information could be used for correlation of end-to-end application performance metrics and system level disk I/O request latencies.

For the approaches in Chapter 5 we recommend the following extensions:

- *Implementation of the time averaged queue lengths and no-interference baseline based policies in an online database installation*. This extension could be implemented as a software admission controller component external to the database application. In an initial training phase the η matrix needs to be built, e.g., using the TPC-H workload. The matrix is subsequently passed as input to the admission controller. In the second phase the controller could intercept incoming TPC-H queries and change the admission order to the database application based on the prioritizations derived from the η matrix.

- *Dynamic construction of the η matrix at runtime instead of during offline training*. This extension would have the advantage of less training overheads. In addition, the η matrix could be automatically extended in presence of changing workload classes. The implementation could initially set all values of the η matrix to a predefined value, e.g., 0. Based on runtime measurements the matrix values could be dynamically updated.

- *Comparative evaluation of admission policy fairness*. Our proposed policies aim to prioritize work intensive query mixes at the expense of other query mixes. A possible extension of our work could be a time-varying analysis of the query prioritizations introduced by our policies. For example, in simulation each query could maintain statistics of favourable and unfavourable prioritization decisions. Such statistics could be beneficial for a comparative evaluation of the fairness of the proposed policies.

A List of Publications

Refereed Papers in Journals

sosym12 — S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, P. Kilpatrick. *Performance Models of Storage Contention in Cloud Environments*, Journal of Software and Systems Modeling (SoSyM), Springer, Mar 2012.

Refereed Papers in Conferences

cloud12 — S. Kraft, G. Casale, A. Jula, P. Kilpatrick, D. Greer. *WIQ: Work-Intensive Query Scheduling for In-Memory Database Systems*, Proc. of IEEE CLOUD, Honolulu, HI, USA, IEEE, Jun 2012.

icpe11 — S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, P. Kilpatrick. *IO Performance Prediction in Consolidated Virtualized Environments*, Proc. of ACM/SPEC ICPE, Karlsruhe, Germany, ACM, Mar 2011.

valuetools09 — S. Kraft, S.Pacheco-Sanchez, G. Casale, S.Dawson. *Estimating Service Resource Consumption From Response Time Measurements*, Proc. of VALUETOOLS, Pisa, Italy, ACM Press, Oct 2009.

Refereed Papers in Workshops

rossa09 — J. Rolia, G. Casale, D. Krishnamurthy, S. Dawson, S. Kraft . *Predictive Modelling of SAP ERP Applications: Challenges and Solutions*, Proc. of ROSSA, Pisa, Italy, ACM Press, Oct 2009.

Invited Papers

dcperf11 — G. Casale, S. Kraft, D. Krishnamurthy. *A Model of Storage I/O Performance Interference in Virtualized Systems*, Proc. of DCPerf Workshop, Minneapolis, MN, USA, IEEE, June 2011.

B List of Patents

06/2012 *Simulation Techniques for Predicting In-Memory Database Systems Performance*, Filed US Patent Application, No. 13/530,847, filed jointly with G. Casale and A. Jula.

03/2011 *Disk Input/Output Response Time Prediction in Consolidated Virtualized Environments*, Filed US Patent Application, No. 13/418,879, filed jointly with G. Casale and D. Krishnamurthy.

09/2010 *A Technique for Estimating Service Resource Consumption From Response Time Measurements*, Filed US Patent Application, No. 12/580,901, filed jointly with G. Casale, S. Dawson, and S. Pacheco-Sanchez.

C Statistics of Benchmark Experiments

Table C.1: Summary Number of Benchmark Experiments, Number of Individual Benchmark Runs, Aggregated Benchmark Run Length, and Number of Analyzed Measurement Samples.

Workload	*Experiments*	*Runs*	*Length (min)*	*Analyzed Samples*
PM-1	8	120	600	31,599,255
PM-2	8	120	600	24,933,654
FFSB-1_S	8	120	600	8,653,807
FFSB-1_R	8	120	600	8,000,687
FFSB-2_S	8	120	600	6,217,162
FFSB-2_R	8	120	600	7,503,867
Sum	**48**	**720**	**3600**	**86,908,432**
File	13	130	650	14,768,374
Mail	14	140	700	21,660,664
Web	15	150	750	16,744,931
Sum	**42**	**420**	**2100**	**53,173,969**
TOTAL	**90**	**1140**	**5700 (95h)**	**140,082,401**

D Benchmark Configuration Files

PM-1

```
# working directory
set location /home/more/postmark/bm
# number of files initially created
set number 10000
# number of create/delete and read/append transactions
set transactions 250000
set report verbose
run
quit
```

PM-2

```
# working directory
set location /home/more/postmark/bm
# number of files initially created
set number 10000
# number of create/delete and read/append transactions
set transactions 250000
# file size range in bytes
set size 10000 20000
# size of reads in bytes
set read 2048
# size of writes in bytes
set write 2048
set report verbose
run
quit
```

FFSB-1_S

```
# global options
# 0: dont use direct IO
directio = 0
# time in sec
time = 300

[filesystem]
 location=/home/more/ffsb/bm
 num_files       = 400
 num_dirs        = 10
 size_weight 4k 33
 size_weight 8k 21
 size_weight 16k 13
 size_weight 32k 10
 size_weight 64k 8
 size_weight 128k 5
 size_weight 256k 4
 size_weight 512k 3
 size_weight 8m 2
 size_weight 32m 1
 init_size = 100m
 reuse = 0
 agefs = 0
 [threadgroup]
  num_threads = 10
  write_size = 400
  write_blocksize = 1024
  create_weight = 10
  append_weight = 10
  delete_weight = 1
 [end]
 desired_util = 0.2
[end]
[threadgroup]
 num_threads = 100
 append_weight = 1
 append_fsync_weight = 1
 stat_weight = 1
 write_weight = 10
 read_weight = 10
 create_weight = 1
 create_fsync_weight = 1
 delete_weight = 1
 readall_weight = 1
 writeall_weight = 1
 writeall_fsync_weight = 1
 open_close_weight = 1
 read_random = 0
 write_random = 0
 write_size = 4k
 write_blocksize = 4k
 read_size = 4k
 read_blocksize = 4k
 #think time in ms
 op_delay = 0
 [stats]
   enable_stats = 1
 enable_range = 0
 msec_range 0.00 0.01
 msec_range 0.01 0.02
 msec_range 0.02 0.03
 msec_range 0.03 0.04
 msec_range 0.04 0.05
 msec_range 0.05 0.1
 msec_range 0.1 0.2
 msec_range 0.2 0.5
 msec_range 0.5 1.0
 msec_range 1.0 2.0
 msec_range 2.0 3.0
 msec_range 3.0 4.0
 msec_range 4.0 5.0
 msec_range 5.0 10.0
 msec_range 10.0 10000.0
[end]
[end]
```

FFSB-1_R

```
# global options
# 0: dont use direct IO
directio = 0
# time in sec
time = 300

[filesystem]
 location=/home/more/ffsb/bm
 num_files       = 400
 num_dirs        = 10
 size_weight 4k 33
 size_weight 8k 21
 size_weight 16k 13
 size_weight 32k 10
 size_weight 64k 8
 size_weight 128k 5
 size_weight 256k 4
 size_weight 512k 3
 size_weight 8m 2
 size_weight 32m 1
 init_size = 100m
 reuse = 0
 agefs = 0
 [threadgroup]
  num_threads = 10
  write_size = 400
  write_blocksize = 1024
  create_weight = 10
  append_weight = 10
  delete_weight = 1
 [end]
 desired_util = 0.2
[end]
[threadgroup]
 num_threads = 100
 append_weight = 1
 append_fsync_weight = 1
 stat_weight = 1
 write_weight = 10
 read_weight = 10
 create_weight = 1
 create_fsync_weight = 1
 delete_weight = 1
 readall_weight = 1
 writeall_weight = 1
 writeall_fsync_weight = 1
 open_close_weight = 1
 read_random = 1
 write_random = 1
 write_size = 4k
 write_blocksize = 4k
 read_size = 4k
 read_blocksize = 4k
 #think time in ms
 op_delay = 0
 [stats]
   enable_stats = 1
 enable_range = 0
 msec_range 0.00 0.01
 msec_range 0.01 0.02
 msec_range 0.02 0.03
 msec_range 0.03 0.04
 msec_range 0.04 0.05
 msec_range 0.05 0.1
 msec_range 0.1 0.2
 msec_range 0.2 0.5
 msec_range 0.5 1.0
 msec_range 1.0 2.0
 msec_range 2.0 3.0
 msec_range 3.0 4.0
 msec_range 4.0 5.0
 msec_range 5.0 10.0
 msec_range 10.0 10000.0
[end]
[end]
```

FFSB-2_S

```
# global options
# 0: dont use direct IO
directio = 0
# time in sec
time = 300

[filesystem]
 location=/home/more/ffsb/bm
 num_files = 400
 num_dirs  = 10
 size_weight 32k 33
 size_weight 64k 21
 size_weight 128k 13
 size_weight 256k 12
 size_weight 512k 11
 size_weight 8m 5
 size_weight 32m 5
 init_size = 100m
 reuse  = 0
 agefs = 0
 [threadgroup]
  num_threads = 10
write_size = 400
write_blocksize = 1024
create_weight = 10
append_weight = 10
delete_weight = 1
 [end]
 desired_util = 0.2
[end]
[threadgroup]
 num_threads  = 100
 append_weight = 1
 append_fsync_weight = 1
 stat_weight = 1
 write_weight = 10
 read_weight = 10
 create_weight = 1
 create_fsync_weight = 1
 delete_weight = 1
 readall_weight = 1
 writeall_weight = 1
 writeall_fsync_weight = 1
 open_close_weight = 1
 read_random = 0
 write_random = 0
 write_size = 32k
 write_blocksize = 4k
 read_size = 32k
 read_blocksize = 4k
 #think time in ms
 op_delay = 0
 [stats]
  enable_stats = 1
enable_range = 0
msec_range 0.00 0.01
msec_range 0.01 0.02
msec_range 0.02 0.03
msec_range 0.03 0.04
msec_range 0.04 0.05
msec_range 0.05 0.1
msec_range 0.1 0.2
msec_range 0.2 0.5
msec_range 0.5 1.0
msec_range 1.0 2.0
msec_range 2.0 3.0
msec_range 3.0 4.0
msec_range 4.0 5.0
msec_range 5.0 10.0
msec_range 10.0 10000.0
 [end]
[end]
```

FFSB-2_R

```
# global options
# 0: dont use direct IO
directio = 0
# time in sec
time = 300

[filesystem]
 location=/home/more/ffsb/bm
 num_files = 400
 num_dirs  = 10
 size_weight 32k 33
 size_weight 64k 21
 size_weight 128k 13
 size_weight 256k 12
 size_weight 512k 11
 size_weight 8m 5
 size_weight 32m 5
 init_size = 100m
 reuse  = 0
 agefs = 0
 [threadgroup]
  num_threads = 10
write_size = 400
write_blocksize = 1024
create_weight = 10
append_weight = 10
delete_weight = 1
 [end]
 desired_util = 0.2
[end]
[threadgroup]
 num_threads  = 100
 append_weight = 1
 append_fsync_weight = 1
 stat_weight = 1
 write_weight = 10
 read_weight = 10
 create_weight = 1
 create_fsync_weight = 1
 delete_weight = 1
 readall_weight = 1
 writeall_weight = 1
 writeall_fsync_weight = 1
 open_close_weight = 1
 read_random = 1
 write_random = 1
 write_size = 32k
 write_blocksize = 4k
 read_size = 32k
 read_blocksize = 4k
 #think time in ms
 op_delay = 0
 [stats]
  enable_stats = 1
enable_range = 0
msec_range 0.00 0.01
msec_range 0.01 0.02
msec_range 0.02 0.03
msec_range 0.03 0.04
msec_range 0.04 0.05
msec_range 0.05 0.1
msec_range 0.1 0.2
msec_range 0.2 0.5
msec_range 0.5 1.0
msec_range 1.0 2.0
msec_range 2.0 3.0
msec_range 3.0 4.0
msec_range 4.0 5.0
msec_range 5.0 10.0
msec_range 10.0 10000.0
 [end]
[end]
```

Filebench Mail

```
load varmail
set $nfiles=50000
set $dir=/home/more/filebench/bm
run 300
```

Filebench Web

```
load webserver
set $nfiles=50000
set $dir=/home/more/filebench/bm
run 300
```

Filebench File

```
load fileserver
set $nfiles=50000
set $dir=/home/more/filebench/bm
run 300
```

References

[1] Blktrace-Linux man page. http://linux.die.net/man/8/blktrace. Last Accessed: 16/11/2012.

[2] DVD store. http://linux.dell.com/dvdstore/. Last Accessed: 16/11/2012.

[3] FFSB-v6.0-rc2. http://sourceforge.net/projects/ffsb. Last Accessed: 16/11/2012.

[4] IBM solidDB-fastest data delivery. http://www-01.ibm.com/software/data/soliddb/. Last Accessed: 16/11/2012.

[5] Kernel based virtual machine. http://www.linux-kvm.org/page/Main_Page. Last Accessed: 16/11/2012.

[6] Openfiler. http://www.openfiler.com. Last Accessed: 16/11/2012.

[7] Oracle and TimesTen. http://www.oracle.com/timesten. Last Accessed: 16/11/2012.

[8] Parallels cloud server 6.0 beta 2. http://www.parallels.com/products/pcs/. Last Accessed: 19/11/2012.

[9] Postmark-1.51-7. http://packages.debian.org/sid/postmark. Last Accessed: 16/11/2012.

[10] SAP. http://www.sap.com. Last Accessed: 16/11/2012.

[11] Storage queues and performance. http://communities.vmware.com/docs/DOC-6490. Last Accessed: 16/11/2012.

[12] TPC-H: Transaction processing performance council. http://www.tpc.org/tpch/default.asp. Last Accessed: 16/11/2012.

[13] TPC-R: Transaction processing performance council. http://www.tpc.org/tpcr/default.asp. Last Accessed: 16/11/2012.

[14] Tshark manpage. http://www.wireshark.org/docs/man-pages/tshark.html. Last Accessed: 16/11/2012.

[15] UMass trace repository. http://traces.cs.umass.edu. Last Accessed: 16/11/2012.

[16] Virident FlashMAX II. http://www.virident.com/products/flashMAX/. Last Accessed: 16/11/2012.

[17] VirtualBox. https://www.virtualbox.org/. Last Accessed: 16/11/2012.

[18] VMware. http://www.vmware.com. Last Accessed: 16/11/2012.

[19] Xen. http://www.xen.org/. Last Accessed: 16/11/2012.

[20] Understanding full virtualization, paravirtualization, and hardware assist. Tech. Rep. Revision 20070911 Item: WP-028-PRD-01-01, VMware, 2007.

[21] Red hat enterprise linux 5 IO tuning guide. Tech. Rep. Revision 1.0, Red Hat Inc., 2008.

[22] Timekeeping in VMware virtual machines. Tech. Rep. Revision:20081017 Item:WP-065-PRD-02-02, VMware, 2008.

[23] IBM DB2 9.7 for linux, UNIX, and windows - query patroller administration and user's guide. Tech. Rep. SC27-2467-00, IBM, 2009.

[24] iSCSI SAN configuration guide. Tech. Rep. Revision: 20090313, Item: EN-000035-01, VMware Inc., 2009.

[25] TMPFS - FreeBSD man pages. http://www.freebsd.org/cgi/man.cgi?query=tmpfs, 2010. Last Accessed: 16/11/2012.

[26] Filebench. http://sourceforge.net/apps/mediawiki/filebench/index.php?title=Main_Page, 2011. Last Accessed: 16/11/2012.

[27] IOTTA repository home. http://iotta.snia.org, 2011. Last Accessed: 16/11/2012.

[28] ABADI, D. J. *Query Execution in Column-Oriented Database Systems*. PhD thesis, Massachusetts Institute of Technology, 2008.

[29] ABBOTT, R., AND GARCIA-MOLINA, H. Scheduling real-time transactions. In *SIGMOD Rec. 17*, 1 (1988), 71–81.

[30] ABBOTT, R., AND GARCIA-MOLINA, H. Scheduling real-time transactions with disk resident data. In *International Conference on Very Large Data Bases, VLDB* (1989), Morgan Kaufmann Publishers Inc.

[31] ABBOTT, R. K., AND GARCIA-MOLINA, H. Scheduling real-time transactions: A performance evaluation. In *TODS 17* (1992), 513–560.

[32] ADAMS, K., AND AGESEN, O. A comparison of software and hardware techniques for x86 virtualization. In *International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS* (2006), ACM.

[33] AGUILERA, M. K., KEETON, K., MERCHANT, A., MUNISWAMY-REDDY, K.-K., AND UYSAL, M. Improving recoverability in multi-tier storage systems. In *International Conference on Dependable Systems and Networks, DSN* (2007), IEEE.

[34] AHMAD, I. Easy and efficient disk I/O workload characterization in VMware ESX server. In *International Symposium on Workload Characterization, IISWC* (2007), IEEE.

[35] AHMAD, I., ANDERSON, J. M., HOLLER, A. M., KAMBO, R., AND MAKHIJA, V. An analysis of disk performance in VMware ESX server virtual machines. In *International Workshop on Workload Characterization, WWC-6* (2003), IEEE.

[36] AHMAD, M., ABOULNAGA, A., AND BABU, S. Query interactions in database workloads. In *International Workshop on Testing Database Systems, DBTest* (2009), ACM.

[37] AHMAD, M., ABOULNAGA, A., BABU, S., AND MUNAGALA, K. Modeling and exploiting query interactions in database systems. In *Conference on Information and Knowledge Management, CIKM* (2008), ACM.

[38] AHMAD, M., ABOULNAGA, A., BABU, S., AND MUNAGALA, K. Qshuffler: Getting the query mix right. In *International Conference on Data Engineering, ICDE* (2008), IEEE.

[39] AHMAD, M., ABOULNAGA, A., BABU, S., AND MUNAGALA, K. Interaction-aware scheduling of report-generation workloads. In *VLDB Journal 20* (2011), 589–615.

[40] AHMAD, M., DUAN, S., ABOULNAGA, A., AND BABU, S. Predicting completion times of batch query workloads using interaction-aware models and simulation. In *International Conference on Extending Database Technology, EDBT* (2011), ACM.

[41] ALBUTIU, M.-C., AND KEMPER, A. Synergy-based workload management. In *International Conference on Very Large Data Bases, VLDB, PhD Workshop* (2009), VLDB Endowment.

[42] ANDERSON, E., HOBBS, M., KEETON, K., SPENCE, S., UYSAL, M., AND VEITCH, A. Hippodrome: Running circles around storage administration. In *Conference on File and Storage Technologies, FAST* (2002), USENIX.

[43] ANSELL, P. S., GLAZEBROOK, K. D., AND KIRKBRIDE, C. Generalised "join the shortest queue" policies for the dynamic routing of jobs to multiclass queues. In *JORS 54*, 4 (2003), 379–389.

[44] AULBACH, S., GRUST, T., JACOBS, D., KEMPER, A., AND RITTINGER, J. Multi-tenant databases for software as a service: Schema-mapping techniques. In *International Conference on Management of Data, SIGMOD* (2008), ACM.

[45] AXBOE, J. Linux block IO - present and future. In *Linux Symposium* (2004).

[46] BALSAMO, S. Product form queueing networks. In *Performance Evaluation, PEVA* (2000), Springer.

[47] BALSAMO, S., MARCO, A. D., INVERARDI, P., AND SIMEONI, M. Model-based performance prediction in software development: A survey. In *TSE 30*, 5 (2004), 295–310.

[48] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Symposium on Operating Systems Principles, SOSP* (2003), ACM.

[49] BASKETT, F., CHANDY, K. M., MUNTZ, R. R., AND PALACIOS, F. G. Open, closed, and mixed networks of queues with different classes of customers. In *J. ACM 22*, 2 (1975), 248–260.

[50] BEGIN, T., BRANDWAJN, A., BAYNAT, B., WOLFINGER, B. E., AND FDIDA, S. High-level approach to modeling of observed system behavior. In *PEVA 67*, 5 (2010), 386–405.

[51] BENEVENUTO, F., FERNANDES, C., SANTOS, M., ALMEIDA, V., ALMEIDA, J., JANAKIRAMAN, G., AND SANTOS, J. Performance models for virtualized applications. In *International Symposium on Parallel and Distributed Processing with Applications, ISPA*, vol. 4331 of *LNCS*. 2006.

[52] BENNANI, M. N., AND MENASCÉ, D. A. Resource allocation for autonomic data centers using analytic performance models. In *International Conference on Autonomic Computing, ICAC* (2005), IEEE.

[53] BENNETT, J. C., AND ZHANG, H. WF^2Q: Worst-case fair weighted fair queueing. In *International Conference on Computer Communications, INFOCOM* (1996), IEEE.

[54] BHOJ, P., RAMANATHAN, S., AND SINGHAL, S. Web2K: Bringing QoS to web servers. Tech. Rep. HPL-2000-61, HP Laboratories Palo Alto, 2000.

[55] BINNIG, C., HILDENBRAND, S., AND FÄRBER, F. Dictionary-based order-preserving string compression for main memory column stores. In *International Conference on Management of Data, SIGMOD* (2009), ACM.

[56] BOLCH, G., GREINER, S., DE MEER, H., AND TRIVEDI, K. S. *Queueing Networks and Markov Chains*, 2nd edition ed. John Wiley and Sons, 2006.

[57] BONOMI, F. On job assignment for a parallel system of processor sharing queues. In *TOC 39*, 7 (1990), 858–869.

[58] BOUTCHER, D., AND CHANDRA, A. Does virtualization make disk scheduling passé? In *SIGOPS OSR 44*, 1 (2010), 20–24.

[59] BROWN, K. P., CAREY, M. J., AND LIVNY, M. Managing memory to meet multiclass workload response time goals. In *International Conference on Very Large Data Bases, VLDB* (1993), Morgan Kaufmann Publishers Inc.

[60] BROWN, K. P., MEHTA, M., CAREY, M. J., AND LIVNY, M. Towards automated performance tuning for complex workloads. In *International Conference on Very Large Data Bases, VLDB* (1994), Morgan Kaufmann Publishers Inc.

[61] BRUNELLE, A. D. blktrace user guide, 2007.

[62] BUZEN, J. P. Fundamental laws of computer system performance. In *Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS* (1976), ACM.

[63] CALZAROSSA, M., AND SERAZZI, G. Workload characterization: A survey. In *Proc. of the IEEE 81*, 8 (1993), 1136 –1150.

[64] CAREY, M. J., JAUHARI, R., AND LIVNY, M. Priority in DBMS resource scheduling. In *International Conference on Very Large Data Bases, VLDB* (1989), Morgan Kaufmann.

[65] CAREY, M. J., KRISHNAMURTHI, S., AND LIVNY, M. Load control for locking: The "half-and-half" approach. In *Symposium on Principles of Database Systems, PODS* (1990), ACM.

[66] CAREY, M. J., AND LIVNY, M. Distributed concurrency control performance: A study of algorithms, distribution, and replication. In *International Conference on Very Large Data Bases, VLDB* (1988), Morgan Kaufmann.

[67] CASALE, G., MI, N., AND SMIRNI, E. Bound analysis of closed queueing networks with workload burstiness. In *Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS* (2008), ACM.

[68] CASALE, G., MI, N., AND SMIRNI, E. CWS: A model-driven scheduling policy for correlated workloads. In *SIGMETRICS PEVA 38*, 1 (2010), 251–262.

[69] CASALE, G., ZHANG, E. Z., AND SMIRNI, E. KPC-toolbox: Simple yet effective trace fitting using markovian arrival processes. In *International Conference on the Quantitative Evaluation of Systems, QEST* (2008), IEEE.

[70] CHA, S. K., AND SONG, C. P*TIME: Highly scalable OLTP DBMS for managing update-intensive stream workload. In *International Conference on Very Large Data Bases, VLDB* (2004), Morgan Kaufmann.

[71] CHANDRA, A., GONG, W., AND SHENOY, P. Dynamic resource allocation for shared data centers using online measurements. In *International Workshop on Quality of Service, IWQoS* (2003), Springer.

[72] CHEN, P. M., LEE, E. K., GIBSON, G. A., KATZ, R. H., AND PATTERSON, D. A. RAID: High-performance, reliable secondary storage. In *ACM Computing Surveys 26*, 2 (1994), 145–185.

[73] CHERKASOVA, L., AND GARDNER, R. Measuring CPU overhead for I/O processing in the xen virtual machine monitor. In *USENIX Annual Technical Conference, ATEC* (2005), USENIX.

[74] CLARK, T. *Designing Storage Area Networks : A Practical Reference for Implementing Storage Area Networks*, 2nd ed. Pearson, 2003.

[75] COFFMAN JR., E. G., AND KLEINROCK, L. Computer scheduling methods and their countermeasures. In *American Federation of Information Processing Societies, AFIPS* (1968), ACM.

[76] COPELAND, G. P., AND KHOSHAFIAN, S. N. A decomposition storage model. In *International Conference on Management of Data, SIGMOD* (1985), ACM.

[77] CREASY, R. J. The origin of the VM/370 time-sharing system. In *IBM Journal of Research and Development 25*, 5 (1981), 483–490.

[78] CROVELLA, M. E., FRANGIOSO, R., AND HARCHOL-BALTER, M. Connection scheduling in web servers. In *USENIX Symposium on Internet Technologies and Systems, USITS* (1999), USENIX.

[79] DEMERS, A., KESHAV, S., AND SHENKER, S. Analysis and simulation of a fair queueing algorithm. In *Symposium Proceedings on Communications Architectures & Protocols, SIGCOMM* (1989), ACM.

[80] DENNING, P. J. Thrashing: Its causes and prevention. In *American Federation of Information Processing Societies, AFIPS* (1968), ACM.

[81] DENNING, P. J., AND BUZEN, J. P. The operational analysis of queueing network models. In *ACM Comput. Surv. 10*, 3 (1978), 225–261.

[82] DEWITT, D. J., KATZ, R. H., OLKEN, F., SHAPIRO, L. D., STONEBRAKER, M. R., AND WOOD, D. A. Implementation techniques for main memory database systems. In *SIGMOD Rec. 14*, 2 (1984), 1–8.

[83] ELEFTHERIOU, E., HAAS, R., JELITTO, J., LANTZ, M. A., AND POZIDIS, H. Trends in storage technologies. In *IEEE Data Eng. Bull. 33*, 4 (2010), 4–13.

[84] ELNIKETY, S., NAHUM, E., TRACEY, J., AND ZWAENEPOEL, W. A method for transparent admission control and request scheduling in e-commerce web sites. In *International World Wide Web Conference, WWW* (2004), ACM.

[85] ERMOLINSKIY, A., AND TEWARI, R. C2Cfs: A collective caching architecture for distributed file access. In *International Conference on High Performance Computing and Communications, HPCC* (2009), IEEE.

[86] FÄRBER, F., CHA, S. K., PRIMSCH, J., BORNHÖVD, C., SIGG, S., AND LEHNER, W. SAP HANA database: Data management for modern business applications. In *SIGMOD Record 40*, 4 (2011), 45–51.

[87] FIELD, T. JINQS: An extensible library for simulating multiclass queueing networks, v1.0 user guide. `http://www.doc.ic.ac.uk/~ajf/Research/manual.pdf`. Last Accessed: 16/11/2012.

[88] FRANK, H. Analysis and optimization of disk storage devices for time-sharing systems. In *J. ACM 16*, 4 (1969), 602–620.

[89] FREITAS, R. F., AND WILCKE, W. W. Storage-class memory: The next storage system technology. In *IBM J. Res. Dev. 52*, 4 (jul 2008), 439–447.

[90] GANGER, G. R. Generating representative synthetic workloads: An unsolved problem. In *CMG Conference* (1995).

[91] GANGER, G. R., AND PATT, Y. N. The process-flow model: Examining I/O performance from the system's point of view. In *Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS* (1993), ACM.

[92] GANGER, G. R., AND PATT, Y. N. Using system-level models to evaluate I/O subsystem designs. In *IEEE TOC 47*, 6 (1998), 667–678.

[93] GNIADY, C., BUTT, A. R., AND HU, Y. C. Program-counter-based pattern classification in buffer caching. In *Symposium on Operating System Design and Implementation, OSDI* (2004), USENIX.

[94] GOLDBERG, R. P. Survey of virtual machine research. In *IEEE Computer Magazine 7*, 6 (1974), 34 –45.

[95] GOLESTANI, S. J. A self-clocked fair queueing scheme for broadband applications. In *International Conference on Computer Communications, INFOCOM* (1994), IEEE.

[96] GÓMEZ, M. E., AND SANTONJA, V. Self-similarity in I/O workload: Analysis and modeling. In *International Workshop on Workload Characterization, WWC* (1998), IEEE.

[97] GÓMEZ, M. E., AND SANTONJA, V. Analysis of self-similarity in I/O workload using structural modeling. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS* (1999), IEEE.

[98] GOYAL, P., VIN, H. M., AND CHEN, H. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. In *Symposium Proceedings on Communications Architectures & Protocols, SIGCOMM* (1996), ACM.

[99] GULATI, A., AND AHMAD, I. Towards distributed storage resource management using flow control. In *SIGOPS Oper. Syst. Rev. 42*, 6 (2008), 10–16.

[100] GULATI, A., AHMAD, I., AND WALDSPURGER, C. A. PARDA: Proportional allocation of resources for distributed storage access. In *Conference on File and Storage Technologies, FAST* (2009), USENIX.

[101] GULATI, A., KUMAR, C., AND AHMAD, I. Storage workload characterization and consolidation in virtualized environments. In *International Workshop on Virtualization Performance: Analysis, Characterization, and Tools, VPACT* (2009).

[102] GULATI, A., KUMAR, C., AHMAD, I., AND KUMAR, K. BASIL: Automated IO load balancing across storage devices. In *Conference on File and Storage Technologies, FAST* (2010), USENIX.

[103] GULATI, A., MERCHANT, A., AND VARMAN, P. J. pClock: An arrival curve based approach for QoS guarantees in shared storage systems. In *Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS* (2007), ACM.

[104] GULATI, A., MERCHANT, A., AND VARMAN, P. J. mClock: Handling throughput variability for hypervisor io scheduling. In *Symposium on Operating System Design and Implementation, OSDI* (2010), USENIX.

[105] GULATI, A., NAIK, M., AND TEWARI, R. Nache: Design and implementation of a caching proxy for NFSv4. In *Conference on File and Storage Technologies, FAST* (2007), USENIX.

[106] GUPTA, V., HARCHOL-BALTER, M., SIGMAN, K., AND WHITT, W. Analysis of join-the-shortest-queue routing for web server farms. In *Performance Evaluation, PEVA 64*, 9-12 (2007), 1062–1081.

[107] HARCHOL-BALTER, M. Task assignment with unknown duration. In *J. ACM 49*, 2 (2002), 260–288.

[108] HEIDELBERGER, P., AND LAVENBERG, S. S. Computer performance evaluation methodology. In *TOC 33*, 12 (1984), 1195–1220.

[109] HEISS, H.-U., AND WAGNER, R. Adaptive load control in transaction processing systems. In *International Conference on Very Large Data Bases, VLDB* (1991), Morgan Kaufmann Publishers Inc.

[110] HILLSTON, J., HERMANNS, H., HERZOG, U., MERTSIOTAKIS, V., AND RETTELBACH, M. Stochastic process algebras: Integrating qualitative and quantitative modelling. In *International Conference on Formal Description Techniques, FORTE* (1994), Chapman & Hall.

[111] HORDIJK, A., AND KOOLE, G. On the optimality of the generalized shortest queue policy. In *Probability in The Engineering and Informational Sciences 4* (1990), 477–487.

[112] IBARAKI, T., KAMEDA, T., AND KATOH, N. Cautious transaction schedulers for database concurrency control. In *TSE 14*, 7 (1988), 997–1009.

[113] JAIN, R. *The Art of Computer System Performance Analysis*. John Wiley & Sons, 1991.

[114] JIN, W., CHASE, J. S., AND KAUR, J. Interposed proportional sharing for a storage service utility. In *ACM PEVA 32*, 1 (2004), 37–48.

[115] JOUKOV, N., WONG, T., AND ZADOK, E. Accurate and efficient replaying of file system traces. In *Conference on File and Storage Technologies, FAST* (2005), USENIX.

[116] JULA, A. Multicore scalability and NUMA effects on HDB with SAP-H data and TPC-H queries. Tech. rep., SAP, 2011.

[117] JUNG, G., JOSHI, K. R., HILTUNEN, M. A., SCHLICHTING, R. D., AND PU, C. Generating adaptation policies for multi-tier applications in consolidated server environments. In *International Conference on Autonomic Computing, ICAC* (2008), IEEE.

[118] JUNG, G., JOSHI, K. R., HILTUNEN, M. A., SCHLICHTING, R. D., AND PU, C. A cost-sensitive adaptation engine for server consolidation of multitier applications. In *International Middleware Conference, Middleware* (2009), Springer.

[119] KATCHER, J. Postmark: A new file system benchmark. Tech. Rep. 3022, Network Appliance Inc., 1997.

[120] KATOH, N., IBARAKI, T., AND KAMEDA, T. Cautious transaction schedulers with admission control. In *TODS 10*, 2 (1985), 205–229.

[121] KATZ, R. H. High-performance network and channel-based storage. In *Proc. of the IEEE 80*, 8 (1992), 1238 –1261.

[122] KEMPER, A., AND NEUMANN, T. HyPer: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots. In *International Conference on Data Engineering, ICDE* (2011), IEEE.

[123] KESAVAN, M., GAVRILOVSKA, A., AND SCHWAN, K. On disk I/O scheduling in virtual machines. In *Workshop on I/O Virtualization, WIOV* (2010), USENIX.

[124] KOH, Y., KNAUERHASE, R., BRETT, P., BOWMAN, M., WEN, Z., AND PU, C. An analysis of performance interference effects in virtual environments. In *International Symposium on Performance Analysis of Systems and Software, ISPASS* (2007), IEEE.

[125] KRAFT, S., CASALE, G., JULA, A., KILPATRICK, P., AND GREER, D. WIQ: Work-intensive query scheduling for in-memory database systems. In *International Conference on Cloud Computing, CLOUD* (2012), IEEE.

[126] KRAFT, S., PACHECO-SANCHEZ, S., CASALE, G., AND DAWSON, S. Estimating service resource consumption from response time measurements. In *International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS* (2009), ICST.

[127] KUNDU, S., RANGASWAMI, R., DUTTA, K., AND ZHAO, M. Application performance modeling in a virtualized environment. In *International Conference on High-Performance Computer Architecture, HPCA* (2010), IEEE.

[128] LAMPSON, B. W. A scheduling philosophy for multiprocessing systems. In *Commun. ACM 11* (1968), 347–360.

[129] LATOUCHE, G., AND RAMASWAMI, V. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, 1999.

[130] LAZOWSKA, E. D., ZAHORJAN, J., GRAHAM, G. S., AND SEVCIK, K. C. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.

[131] LEGLER, T., LEHNER, W., AND ROSS, A. Data mining with the SAP NetWeaver BI accelerator. In *International Conference on Very Large Data Bases, VLDB* (2006), ACM.

[132] LI, J., CHINNECK, J., WOODSIDE, M., LITOIU, M., AND ISZLAI, G. Performance model driven QoS guarantees and optimization in clouds. In *International Conference on Cloud Computing, CLOUD* (2009), IEEE.

[133] LI, Y., LI, W., AND JIANG, C. A survey of virtual machine system: Current technology and future trends. In *International Symposium on Electronic Commerce and Security, ISECS* (2010), IEEE.

[134] LONG, J. *Storage Networking Protocol Fundamentals*. Cisco Press, 2006.

[135] MAGOUTIS, K., ADDETIA, S., FEDOROVA, A., SELTZER, M. I., CHASE, J. S., GALLATIN, A. J., KISLEY, R., WICKREMESINGHE, R., AND GABBER, E. Structure and performance of the direct access file system. In *USENIX Annual Technical Conference, ATEC* (2002), USENIX.

[136] MANEGOLD, S. *Understanding, Modeling, and Improving Main-Memory Database Performance*. PhD thesis, University of Amsterdam, 2002.

[137] MANEGOLD, S., KERSTEN, M. L., AND BONCZ, P. Database architecture evolution: Mammals flourished long before dinosaurs became extinct. In *Proc. VLDB Endow. 2*, 2 (2009), 1648–1653.

[138] MARSAN, M. A., BALBO, G., CONTE, G., DONATELLI, S., AND FRANCESCHINIS, G. *Modelling with Generalized Stochastic Petri Nets*. Wiley, 1995.

[139] MCDOUGALL, R., AND ANDERSON, J. Virtualization performance: Perspectives and challenges ahead. In *SIGOPS Oper. Syst. Rev. 44* (2010), 40–56.

[140] MCWHERTER, D. T., SCHROEDER, B., AILAMAKI, A., AND HARCHOL-BALTER, M. Priority mechanisms for OLTP and transactional web applications. In *International Conference on Data Engineering, ICDE* (2004), IEEE.

[141] MCWHERTER, D. T., SCHROEDER, B., AILAMAKI, A., AND HARCHOL-BALTER, M. Improving preemptive prioritization via statistical characterization of OLTP locking. In *International Conference on Data Engineering, ICDE* (2005), IEEE.

[142] MEHTA, A., GUPTA, C., AND DAYAL, U. BI batch manager: A system for managing batch workloads on enterprise data-warehouses. In *International Conference on Extending Database Technology, EDBT* (2008), ACM.

[143] MEI, Y., LIU, L., PU, X., AND SIVATHANU, S. Performance measurements and analysis of network I/O applications in virtualized cloud. In *International Conference on Cloud Computing, CLOUD* (2010), IEEE.

[144] MENASCÉ, D. A., AND BENNANI, M. N. Autonomic virtualized environments. In *International Conference on Autonomic and Autonomous Systems, ICAS* (2006), IEEE.

[145] MENG, X., ISCI, C., KEPHART, J., ZHANG, L., BOUILLET, E., AND PENDARAKIS, D. Efficient resource provisioning in compute clouds via VMmultiplexing. In *International Conference on Autonomic Computing, ICAC* (2010), ACM.

[146] MI, N., CASALE, G., CHERKASOVA, L., AND SMIRNI, E. Sizing multi-tier systems with temporal dependence: benchmarks and analytic models. In *JISA 1* (2010), 117–134.

[147] MI, N., ZHANG, Q., RISKA, A., SMIRNI, E., AND RIEDEL, E. Performance impacts of autocorrelated flows in multi-tiered systems. In *PEVA 64*, 9-12 (2007), 1082–1101.

[148] MISHRA, A. K., HELLERSTEIN, J. L., CIRNE, W., AND DAS, C. R. Towards characterizing cloud backend workloads: Insights from google compute clusters. In *SIGMETRICS PEVA 37*, 4 (2010), 34–41.

[149] MÖHRING, R. H., SCHULZ, A. S., AND UETZ, M. Approximation in stochastic scheduling: The power of LP-based priority policies. In *J. ACM 46*, 6 (1999), 924–942.

[150] MOKDAD, L., AND BEN-OTHMAN, J. Admission control mechanism and performance analysis based on stochastic automata networks formalism. In *Journal of Parallel and Distributed Computing 71*, 4 (2011), 594–602.

[151] MÖNKEBERG, A., AND WEIKUM, G. Performance evaluation of an adaptive and robust load control method for the avoidance of data-contention thrashing. In *International Conference on Very Large Data Bases, VLDB* (1992), Morgan Kaufmann Publishers Inc.

[152] MORRIS, R. J. T., AND TRUSKOWSKI, B. J. The evolution of storage systems. In *IBM Systems Journal 42*, 2 (2003), 205–217.

[153] MUNISWAMY-REDDY, K.-K., WRIGHT, C. P., HIMMER, A., AND ZADOK, E. A versatile and user-oriented versioning file system. In *Conference on File and Storage Technologies, FAST* (2004), USENIX.

[154] NAKAMURA, F., YOSHIDA, I., AND KONDO, H. A simulation model for data base system performance evaluation. In *American Federation of Information Processing Societies, AFIPS* (1975), ACM.

[155] NEUTS, M. A versatile markovian point process. In *Journal of Applied Probability 16*, 4 (1979), 764–779.

[156] NI, L. M., AND HWANG, K. Optimal load balancing in a multiple processor system with many job classes. In *TSE 11*, 5 (1985), 491–496.

[157] O'GORMAN, K., ABBADI, A. E., AND AGRAWAL, D. Multiple query optimization in middleware using query teamwork. In *Software-practice & Experience 35*, 4 (2005), 361–391.

[158] ONGARO, D., COX, A. L., AND RIXNER, S. Scheduling I/O in virtual machine monitors. In *International Conference on Virtual Execution Environments, VEE* (2008), ACM.

[159] OZMEN, O., SALEM, K., UYSAL, M., AND ATTAR, M. H. S. Storage workload estimation for database management systems. In *International Conference on Management of Data, SIGMOD* (2007), ACM.

[160] PACHECO-SANCHEZ, S., CASALE, G., SCOTNEY, B., MCCLEAN, S., PARR, G., AND DAWSON, S. Markovian workload characterization for QoS prediction in the cloud. In *International Conference on Cloud Computing, CLOUD* (2011), IEEE.

[161] PADALA, P., ZHU, X., WANG, Z., SINGHAL, S., AND SHIN, K. G. Performance evaluation of virtualization technologies for server consolidation. Tech. Rep. HPL-2007-59, HP Laboratories Palo Alto, 2007.

[162] PAREKH, A. K., AND GALLAGER, R. G. A generalized processor sharing approach to flow control in integrated services networks: The single-node case. In *TON 1*, 3 (1993), 344–357.

[163] PARK, J., AND SEGEV, A. Using common subexpressions to optimize multiple queries. In *International Conference on Data Engineering, ICDE* (1988), IEEE.

[164] PLATTNER, H. A common database approach for OLTP and OLAP using an in-memory column database. In *International Conference on Management of Data, SIGMOD* (2009), ACM.

[165] PU, X., LIU, L., MEI, Y., SIVATHANU, S., KOH, Y., AND PU, C. Understanding performance interference of I/O workload in virtualized cloud environments. In *International Conference on Cloud Computing, CLOUD* (2010), IEEE.

[166] RAATIKAINEN, K. E. E. Cluster analysis and workload classification. In *SIGMETRICS PEVA 20*, 4 (1993), 24–30.

[167] RADKOV, P., YIN, L., GOYAL, P., SARKAR, P., AND SHENOY, P. A performance comparison of NFS and iSCSI for IP-networked storage. In *Conference on File and Storage Technologies, FAST* (2004), USENIX.

[168] REISER, M., AND LAVENBERG, S. S. Mean-value analysis of closed multichain queuing networks. In *J. ACM 27*, 2 (1980), 313–322.

[169] RISKA, A., AND RIEDEL, E. Disk drive level workload characterization. In *USENIX Annual Technical Conference* (2006), USENIX.

[170] ROY, P., SESHADRI, S., SUDARSHAN, S., AND BHOBE, S. Efficient and extensible algorithms for multi query optimization. In *SIGMOD Rec. 29* (2000), 249–260.

[171] RUEMMLER, C., AND WILKES, J. UNIX disk access patterns. In *USENIX Winter Technical Conference* (1993), USENIX.

[172] SADRE, R. *Decomposition-Based Analysis of Queueing Networks*. PhD thesis, University of Twente, 2006.

[173] SAHOO, J., MOHAPATRA, S., AND LATH, R. Virtualization: A survey on concepts, taxonomy and associated security issues. In *International Conference on Computer and Network Technology, ICCNT* (2010), IEEE.

[174] SARKAR, P., VORUGANTI, K., METH, K., BIRAN, O., AND SATRAN, J. Internet protocol storage area networks. In *IBM Systems Journal 42*, 2 (2003), 218 –231.

[175] SCHAFFNER, J., BOG, A., KRÜGER, J., AND ZEIER, A. A hybrid row-column OLTP database architecture for operational reporting. In *Informal Proceedings of the Second International Workshop on Business Intelligence for the Real-Time Enterprise, BIRTE* (2008), Springer.

[176] SCHAFFNER, J., ECKART, B., SCHWARZ, C., BRUNNERT, J., JACOBS, D., AND ZEIER, A. Towards analytics-as-a-service using an in-memory column database. vol. 74 of *Lecture Notes in Business Information Processing, LNBIP*. Springer, 2011, pp. 257–282.

[177] SCHROEDER, B., HARCHOL-BALTER, M., IYENGAR, A., AND NAHUM, E. Achieving class-based QoS for transactional workloads. In *International Conference on Data Engineering, ICDE* (2006), IEEE.

[178] SCHROEDER, B., HARCHOL-BALTER, M., IYENGAR, A., NAHUM, E., AND WIERMAN, A. How to determine a good multi-programming level for external scheduling. In *International Conference on Data Engineering, ICDE* (2006), IEEE.

[179] SEELAM, S. R., ROMERO, R., AND TELLER, P. J. Enhancements to linux I/O scheduling. In *Linux Symposium* (2005).

[180] SEELAM, S. R., AND TELLER, P. J. Virtual I/O scheduler: A scheduler of schedulers for performance virtualization. In *International Conference on Virtual Execution Environments, VEE* (2007), ACM.

[181] SEHGAL, P., TARASOV, V., AND ZADOK, E. Evaluating performance and energy in file system server workloads. In *Conference on File and Storage Technologies, FAST* (2010), USENIX.

[182] SELLIS, T. K. Multiple-query optimization. In *TODS 13*, 1 (1988), 23–52.

[183] SHARMA, A. B., BHAGWAN, R., CHOUDHURY, M., GOLUBCHIK, L., GOVINDAN, R., AND VOELKER, G. M. Automatic request categorization in internet services. In *SIGMETRICS PEVA 36*, 2 (2008), 16–25.

[184] SHRIVER, E., MERCHANT, A., AND WILKES, J. An analytic behavior model for disk drives with readahead caches and request reordering. In *PEVA 26* (1998), 182–191.

[185] SIVATHANU, G., SUNDARARAMAN, S., AND ZADOK, E. Type-safe disks. In *Symposium on Operating System Design and Implementation, OSDI* (2006), USENIX.

[186] SOUNDARARAJAN, V., AND ANDERSON, J. M. The impact of management operations on the virtualized datacenter. In *International Symposium on Computer Architecture, ISCA* (2010), ACM.

[187] TANDRA, R., HEMACHANDRA, N., AND MANJUNATH, D. Join minimum cost queue for multiclass customers: Stability and performance bounds. In *Probability in the Engineering and Informational Sciences 18*, 4 (2004), 445–472.

[188] TANENBAUM, A. S. *Modern Operating Systems*. Prentice Hall, 2001.

[189] TARASOV, V., BHANAGE, S., ZADOK, E., AND SELTZER, M. Benchmarking file system benchmarking: It *IS* rocket science. In *Workshop on Hot Topics in Operating Systems, HotOS* (2011), USENIX.

[190] TRAEGER, A., ZADOK, E., JOUKOV, N., AND WRIGHT, C. P. A nine year study of file system and storage benchmarking. In *TOS 4* (2008), 5:1–5:56.

[191] TRIVEDI, K. S. Analytic modeling of computer systems. In *Computer 11*, 10 (1978), 38–56.

[192] ULUSOY, Ö., AND BELFORD, G. G. A simulation model for distributed real-time database systems. In *Annual Simulation Symposium, ANSS* (1992), IEEE.

[193] VARKI, E. Mean value technique for closed fork-join networks. In *Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS* (1999), ACM.

[194] VARKI, E., AND DOWDY, L. W. Analysis of balanced fork-join queueing networks. In *Joint International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS* (1996), ACM.

[195] VARKI, E., AND DOWDY, L. W. Response time analysis of two server fork-join systems. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS* (1996), IEEE.

[196] VARKI, E., MERCHANT, A., XU, J., AND QIU, X. An integrated performance model of disk arrays. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS* (2003), IEEE.

[197] VOIGT, T., TEWARI, R., FREIMUTH, D., AND MEHRA, A. Kernel mechanisms for service differentiation in overloaded web servers. In *USENIX Annual Technical Conference* (2001), USENIX.

[198] WACHS, M., ABD-EL-MALEK, M., THERESKA, E., AND GANGER, G. R. Argon: Performance insulation for shared storage servers. In *Conference on File and Storage Technologies, FAST* (2007), USENIX.

[199] WANG, M., AILAMAKI, A., AND FALOUTSOS, C. Capturing the spatio-temporal behavior of real traffic data. In *PEVA 49* (2002), 147–163.

[200] WANG, M., AU, K., AILAMAKI, A., BROCKWELL, A., FALOUTSOS, C., AND GANGER, G. R. Storage device performance prediction with CART models. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS* (2004), IEEE.

[201] WANG, M., AU, K., AILAMAKI, A., BROCKWELL, A., FALOUTSOS, C., AND GANGER, G. R. Storage device performance prediction with CART models. Tech. Rep. CMU-PDL-04-103, Parallel Data Laboratory - Carnegie Mellon University, 2004.

[202] WANG, Y., AND MERCHANT, A. Proportional-share scheduling for distributed storage systems. In *Conference on File and Storage Technologies, FAST* (2007), USENIX.

[203] WANG, Z., ZHU, X., PADALA, P., AND SINGHAL, S. Capacity and performance overhead in dynamic resource allocation to virtual containers. In *IFIP/IEEE International Symposium on Integrated Network Management, IM* (2007), IEEE.

[204] WHITT, W. Deciding which queue to join: Some counterexamples. In *Operations Research 34*, 1 (1986), 55–62.

[205] WIERMAN, A., AND HARCHOL-BALTER, M. Classifying scheduling policies with respect to unfairness in an M/GI/1. In *SIGMETRICS PEVA 31*, 1 (2003), 238–249.

[206] WINSTON, W. Optimality of the shortest line discipline. In *Journal of Applied Probability 14*, 1 (1977), 181–189.

[207] WOOD, T., CHERKASOVA, L., OZONAT, K., AND SHENOY, P. Profiling and modeling resource usage of virtualized applications. In *International Middleware Conference, Middleware* (2008), Springer.

[208] YU, P. S., BALSAMO, S., AND LEE, Y.-H. Dynamic transaction routing in distributed database systems. In *TSE 14*, 9 (1988), 1307–1318.

[209] ZHANG, G., CHIU, L., AND LIU, L. Adaptive data migration in multi-tiered storage based cloud environment. In *International Conference on Cloud Computing, CLOUD* (2010), IEEE.

[210] ZHU, N., CHEN, J., AND CHIUEH, T.-C. TBBT: Scalable and accurate trace replay for file server evaluation. In *Conference on File and Storage Technologies, FAST* (2005), USENIX.

[211] ZICHEN, X., YI-CHENG, T., AND XIAORUI, W. Exploring power-performance tradeoffs in database systems. In *International Conference on Data Engineering, ICDE* (2010), IEEE.

Index