Automated Workload Characterization for I/O Performance Analysis in Virtualized Environments

Axel Busch*, Qais Noorshams*, Samuel Kounev†,
Anne Koziolek*, Ralf Reussner*, Erich Amrehn‡

*Karlsruhe Institute of Technology, Karlsruhe, Germany (email: [lastname]@kit.edu)
†University of Würzburg, Würzburg, Germany (email: samuel.kounev@uni-wuerzburg.de)
‡IBM Research & Development, Böblingen, Germany (email: amrehn@de.ibm.com)

ABSTRACT

Next generation IT infrastructures are highly driven by virtualization technology. The latter enables flexible and efficient resource sharing allowing to improve system agility and reduce costs for IT services. Due to the sharing of resources and the increasing requirements of modern applications on I/O processing, the performance of storage systems is becoming a crucial factor. In particular, when migrating or consolidating different applications the impact on their performance behavior is often an open question. Performance modeling approaches help to answer such questions, a prerequisite, however, is to find an appropriate workload characterization that is both easy to obtain from applications as well as sufficient to capture the important characteristics of the application. In this paper, we present an automated workload characterization approach that extracts a workload model to represent the main aspects of I/O-intensive applications using relevant workload parameters, e.g., request size, read-/write ratio, in virtualized environments. Once extracted, workload models can be used to emulate the workload performance behavior in real-world scenarios like migration and consolidation scenarios. We demonstrate our approach in the context of two case studies of representative system environments. We present an in-depth evaluation of our workload characterization approach showing its effectiveness in workload migration and consolidation scenarios. We use an IBM System z equipped with an IBM DS8700 and a Sun Fire system as state-of-the-art virtualized environments. Overall, the evaluation of our workload characterization approach shows promising results to capture the relevant factors of I/O-intensive applications.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques, Modeling techniques; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.

ACM 978-1-4503-3248-4/15/01 ...$15.00.
http://dx.doi.org/10.1145/2668930.2688050.

Keywords

I/O; Storage; Workload; Characterization; Virtualized; Performance

1. INTRODUCTION

Today, I/O-intensive applications support major processes of many organizations in their daily business. Workloads as mail servers, file servers, and video servers show highly I/O-intensive workload profiles, cf. [6, 14]. Their huge data volumes require high-performing dedicated external storage infrastructures. To reduce hardware costs, energy consumption, and administration costs, applications are increasingly deployed in virtualized environments. Current forecasts predict a global virtualized server market growth of 31 % until 2016 [20]. At the same time, the amount of stored digital data will double every two years until 2020 [11]. 40 % of this data will be stored or processed in the cloud. Further, a consolidation of dedicated storage systems into a shared storage infrastructure is under way, cf. [28, 21]. Nevertheless, consolidating several applications on one shared infrastructure introduces complex performance implications due to mutual interference.

To allow the consolidation of applications and services while respecting Service Level Agreements (SLAs), predicting the performance implications becomes a crucial factor. Such a prediction, however, requires tailored performance models that in turn require a significant amount of expertise to create the models. Even when such performance modeling approaches are applied, it is unclear which exact workload parameters are required as input since different approaches use different parameters [17, 18, 30]. Furthermore, it is often unclear how to obtain the parameters for given applications and if they are even sufficient to describe them [10].

To address this discrepancy, in this paper, we develop an automated workload characterization approach to extract workload models [16] that are representations of the main aspects of I/O-intensive applications in virtualized environments. Using the relevant workload parameters identified in previous work [23] as basis, we present a formalized and automated workload characterization approach for running I/O-intensive workloads in virtualized environments. We have tailored our approach to enable a non-invasive and lightweight monitoring, yet with a level of abstraction such that the parameters are practically obtainable.

To evaluate our approach, we perform a comprehensive evaluation to demonstrate its workload modeling performance for common business workloads. We present two case studies showing how our approach can be used for performance analysis. In the case studies, we demonstrate how to use the workload description for measurement-based performance predictions in two scenarios. Our first case study is focused on evaluating the quality of the workload characterization. We execute two representative workloads on a state-of-the art IBM mainframe system to show the workload model performance of our approach. The second case study demonstrates how it can be used in migration and consolidation scenarios. To draw conclusions on the performance of a given application, typical approaches involve the installation and setup of the application to be analyzed. The installation sequence of complex applications can be very challenging or even infeasible due to legal constraints. Using our approach, the migration of complex workloads can be conducted in a predictable manner. Once the workload is modeled, a given workload can be emulated on arbitrary hardware. The installation and setup of the actual application under test on the target system is not necessary. Thus, characterizing a workload for performance model generation can be performed with less effort. Two scenarios address this area: In the migration scenario, we show the prediction of the performance behavior of a certain workload on a Sun Fire system by means of the extracted workload data. The second scenario addresses a consolidation scenario that extends the migration scenario by considering two different workloads. In this scenario, we use a second virtual machine to execute both workloads on one machine in parallel. The consolidation scenario demonstrates the ability of our approach to predict the performance behavior of consolidated workloads in a virtualized environment.

In summary, the contribution of this paper is a formalized and fully-automated methodology for workload characterization specifically targeted at I/O-intensive applications in virtualized environments without requiring invasive or proprietary monitoring tools. In contrast to related work, we perform an extensive validation of this methodology: i) We perform a detailed evaluation using representative application workloads to show the effectiveness of our approach in different scenarios with state-of-the-art server hardware. ii) We show that our approach can be used for performance prediction in a migration scenario. iii) Finally, we show the use of the approach in a workload consolidation scenario.

This paper is organized as follows: Section 2 describes our workload characterization approach. We present a formalization of our characterization methodology and introduce the automation of our approach. Section 3 describes the experimental setup, i.e., the systems under study as well as the software environment for our measurements. Section 4 presents the two case studies evaluating our characterization approach as well as migration and consolidation scenarios. Section 5 presents related work. We finish with a conclusion in Section 6.

2. CHARACTERIZATION APPROACH

A high-level view of our workload characterization approach is shown in Figure 1. The high-level workload, e.g., a file upload, is processed by a certain I/O-intensive business application, e.g., a file server system. From a storage point of view, this high-level workload is transformed into a low-level workload comprised of a sequence of read and write requests. This low-level workload is analyzed by our approach. Using monitoring tools allow to extract the requests' properties, i.e., the workload parameters of the model. The workload model is described by a set of metrics that quantify the various workload parameters. In our approach, the workload model is described by a set of storage-specific metrics that quantify the workload. Once extracted, the workload properties are mapped to a reference benchmark. As a reference benchmark, we use the *Flexible File System Benchmark*[1] [1] (FFSB). Using the reference benchmark, we emulate the original low-level workload on the target system.

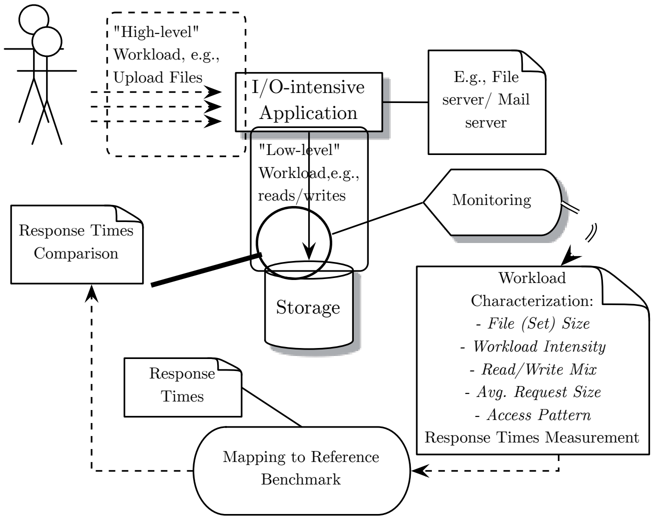


Figure 1: Workload Characterization Approach

A simple capture-replay mechanism would not be sufficient in terms of flexibility. In contrast to a capture-replay mechanism, emulating a workload allows to vary the desired parameters (i.e., file size, workload intensity,. . .). Therefore, different scenarios, e.g., scaling scenarios, become possible.

For validation, we calculate the prediction error between the workload's response times and the reference benchmark. Thus, conclusions on the performance of the extraction sequence become possible.

The contribution of our approach is the formalized and fully-automated characterization methodology for I/O-intensive workloads.

2.1 Workload Characterization

In this section, we describe and formalize the set of workload metrics that we consider for our workload model. In [23], a set of performance-influencing factors in virtualized storage environments was identified. Further, in [22, 24] this set was used as a basis for their performance model. Since our approach is targeted at virtualized systems, we use a subset of these factors for our workload characterization. They respect the limited monitoring possibilities as well as the limited control on system settings. In addition, they represent an adequate level of abstraction for our specific goals.

The set of influencing factors can be divided into two classes, the workload and system factors. In this work, we concentrate on the basic parameters of a workload mix,

[1] http://github.com/FFSB-prime

cf. Figure 2. In the following, we describe our selected set in detail:

- *File size:* Physically allocated space on disk per file. It determines the limits for sequential requests.
- *File set size:* Total physically allocated space on disk. This value influences locality of requests, caching algorithms, and data placement strategies.
- *Workload intensity:* We approximate the workload intensity as the number of threads running in parallel.
- *Request mix:* Proportion between read and write requests.
- *Avg. request size:* Average size of each request processed by the storage system.
- *Request access pattern:* Requests can access data on the disk sequentially or randomly.

In the following, we present the set of metrics we use to measure the performance-influencing factors. Our intention is to monitor the respective parameters over time and to obtain the mean values over the measurement period. Our monitoring tools capture discrete values over time periodically. Thus, we approximate the integral over time to a summation of discrete values used to calculate the mean values.

The *file size* may change significantly over time. Operations like creation of new files, deletion of files from the file set, adding or removing data to existing files influence the file size. Let $[0,T], T > 0$ be the observation period. To capture the changes of the file sizes over time, we propose

$$fileSize^{avg} = \int_0^T \frac{\sum_{\iota=1}^{n(t)} \phi^\iota(t)}{T \cdot n(t)} dt \tag{1}$$

$$= \lim_{|\Delta|\to 0} \sum_{k=1}^{\tau} \frac{\sum_{\iota=1}^{n(x_k)} \phi^\iota(x_k)}{T \cdot n(x_k)} \cdot \Delta t_k \tag{2}$$

$$\approx \frac{1}{\digamma} \sum_{t=1}^{\digamma} \frac{\sum_{\iota=1}^{n(t)} \phi^\iota(t)}{n(t)}, \tag{3}$$

where $\phi^\iota(t)$ is the size of the ι-th file at time t and $n(t)$ is the number of files at time t, $\digamma$ is the number of actual measurement points in the observation period, $\dot{\mathcal{P}} := \{([t_{k-1}, t_k], x_k), 1 \leq k \leq \tau\}$ is a tagged partition of the interval $[0,T]$, i.e., $0 = t_0 \leq x_1 \leq t_1 \leq x_2 \leq \ldots \leq x_\tau \leq t_\tau = T$, $\Delta t_k := t_k - t_{k-1}$, and $|\Delta| := \max_k(\Delta t_k), k \in \{1, \ldots, \tau\}$. In Equation (1) to Equation (3), the integral is transformed to the Riemann sum using $\dot{\mathcal{P}}$ and approximated equidistant points in time.

The *file set size* is a highly changing value in a typical workload life cycle. Thus, again, we propose to measure a set of samples of the file set size over time.

$$fileSetSize^{avg} = \int_0^T \frac{\sum_{\iota=1}^{n(t)} \phi^\iota(t)}{T} dt \tag{4}$$

$$= \lim_{|\Delta|\to 0} \sum_{k=1}^{\tau} \frac{\sum_{\iota=1}^{n(x_k)} \phi^\iota(x_k)}{T} \cdot \Delta t_k \tag{5}$$

$$\approx \frac{1}{\digamma} \sum_{t=1}^{\digamma} \sum_{\iota=1}^{n(t)} \phi^\iota(t) \tag{6}$$

In a typical workload, the number of clients changes over time. We capture the number of clients accessing the system over time to capture the *workload intensity*. The workload intensity, here, can be easily adjusted to use, e.g., the number of requests per second.

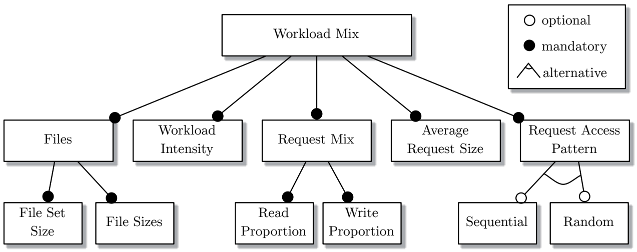


Figure 2: Performance-Influencing Factors (cf. [23])

$$workloadIntensity^{avg} = \int_0^T \frac{\chi(t)}{T} dt \tag{7}$$

$$= \lim_{|\Delta|\to 0} \sum_{k=1}^{\tau} \frac{\chi(x_k)}{T} \cdot \Delta t_k \tag{8}$$

$$\approx \frac{1}{\digamma} \sum_{t=1}^{\digamma} \chi(t), \tag{9}$$

where $\chi(t)$ is the workload intensity (i.e., number of threads, requests per second) at time t.

The *request mix* is captured by the following: Let Γ_0 and Γ_1 be the sets of all observed read and write request sizes:

$$requestMix = \frac{|\Gamma_0|}{|\Gamma_0| + |\Gamma_1|} \tag{10}$$

The *average request size* allows reasoning about the amount of data the storage systems compute in one go. We calculate the average request size for read and write requests as follows:

$$requestSize^{read} = \frac{\sum_{j=0}^{|\Gamma_0|-1} \Gamma_{0,j}}{|\Gamma_0|} \tag{11}$$

$$requestSize^{write} = \frac{\sum_{j=0}^{|\Gamma_1|-1} \Gamma_{1,j}}{|\Gamma_1|} \tag{12}$$

Request access pattern: Data is organized in a fixed length of sequences of bytes, i.e., blocks. When an application requests an amount of data, one or several blocks are accessed in one go. If the accessed blocks of two requests are adjacent, we refer to these requests as sequential, even if they are offset in time. Storage systems often implement algorithms to optimize such sequential requests. They recognize requests whose blocks are consecutive on disk, even if they are interrupted by other requests. This case can easily happen in applications with many execution threads. To detect sequential requests, we propose Algorithm 1, illustrated in Figure 3 and explained below.

Our algorithm determines the percentage of requests that are accessed sequentially. To do so, we search for requests whose block access boundaries are adjacent to each other. The initial observation space comprises all occurred requests from potentially different threads. To distinguish between read and write access patterns, the request space is divided into one read and one write sequence.

Figure 3 illustrates the idea of the algorithm. One request accesses one or several blocks but is depicted as one box, regardless of its size. In the illustration, we depicted sequential accesses using the same pattern. The algorithm compares

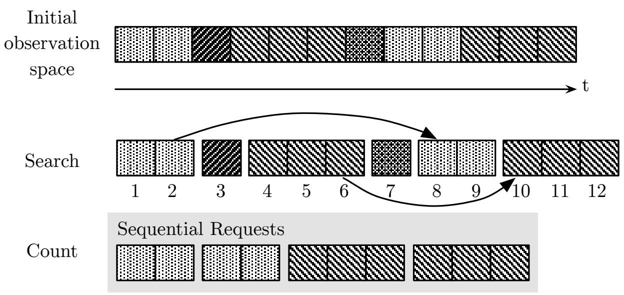


Figure 3: Access Pattern Recognition Algorithm Illustration

each request to all following requests and thus detects that requests 1, 2, 8, and 9 are sequential, as they access consecutive blocks. Additionally, it detects that requests 4, 5, 6, 10, 11, and 12 are sequential.

`Algorithm:` The algorithm's input parameter is a list of n pairs. A pair is defined as $R_i := (block\_start, block\_end)$, where $0 \leq block\_start \leq block\_end$ represents the start and end block number of the i-th request access. The algorithm compares the end block numbers of one request with the start block numbers of the following requests to search for sequential requests. It outputs the proportion of sequential requests in the observation space.

To improve run time and avoid overestimation of sequential requests, we enhanced the algorithm's performance by dividing the observation space into i subsets S_i. Hence, the complexity of our algorithm results in $O(n)$ at best and $O(\omega\lceil\frac{n}{\omega}\rceil^2)$ at worst.

$$S_i := \begin{cases} \{R_{i\cdot\lceil\frac{n}{\omega}\rceil}, \ldots, R_{(i+1)\cdot\lceil\frac{n}{\omega}\rceil-1}\}, & (i+1)\lceil\frac{n}{\omega}\rceil \leq n \\ \{R_{i\cdot\lceil\frac{n}{\omega}\rceil}, \ldots, R_{n-1}\}, & else \end{cases}, \tag{13}$$

where $i \in \{0, \ldots, \omega - 1\}$ and ω is the number of used subsets. The result access pattern ratio is the average of Algorithm 1's result for each subset:

$$accPatternRatio(R) = \frac{\sum_i getAccPat(S_i)}{\omega}$$

$$accPattern(R) = \begin{cases} sequential, & accPatternRatio(R) \geq 0.5 \\ random, & else \end{cases} \tag{14}$$

Our approach avoids overestimation of sequential requests: Sequential requests that are far away from each other in the observation space are not included in the access pattern ratio $accPatternRatio$.

Similar request access pattern heuristics do not respect sequential requests if they are not directly followed up, but interrupted by a non-sequential request, cf. [12]. Our approach respects this and allows a configuration of the observation space using an observation window.

2.2 Characterization Automation

To automate the proposed workload characterization approach, we have extended a tool to automatically execute a workload and obtain its parameters by observing the set of relevant metrics. The tool, called Storage Performance Analyzer (SPA) [3], is a software that supports fully-automatic systematic performance measurements and monitoring of storage system properties. Its architecture allows to analyze arbitrary application workloads. Alternatively, it supports the integration of workload generators, e.g., benchmarks.

Algorithm 1 Access Pattern Recognition Algorithm

```
Configuration:
$R \leftarrow$ Sequence of request pairs
$req \leftarrow$ Number of requests
$req\_seq \leftarrow 0$

Function getAccPat(R):
while $i < req$ do                         // Iterate through requests
    for $j$ such that $i < j < req$ do
        $block\_end = R_{i_2}$              // End block of request $R_i$
        $block\_start = R_{j_1}$            // Start block of request $R_j$
        if $block\_end = block\_start$ then
            $req\_seq \leftarrow req\_seq + 2$   // Count both $R_i, R_j$
            $R \leftarrow R \setminus \{R_i, R_j\}$
            continue while;
        end if
    end for
    $i \leftarrow i + 1$
end while
return $\frac{req\_seq}{req}$
```

Performing manual steps is highly error-prone. SPA supports automatic measurements and therefore allows a coordinated execution of the original workloads. To enable the extraction of workload parameters, the `benchmark controller` was extended to support the synchronized and parallel execution of several workloads and to distinguish these parallel workloads when monitoring. The actual execution, i.e., execute start and stop commands and gathering of log files, of the particular workload is realized by the `benchmark driver` component. For our monitoring goals, we extended the tool architecture by adding a monitor component, cf. class diagram in Figure 4.

The core of the monitor component is the `monitor driver`. It controls the monitoring tools to be prepared, started and stopped on the system under test, as well as processing its measurement values, i.e., extracting a metrics set. The class diagram shows several concrete monitor drivers, e.g., `FilesizeMonitorDriver`, which is an implementation of the abstract monitor driver class. `IndependentVariables` stores the configuration parameters of the monitoring tools. To extract request block relevant metrics, we use Blktrace [7] in SPA.

Blktrace is a block layer I/O tracing tool. Since it collects data on the application layer it is executed on the system under test. Using Blktrace, we obtain detailed disk request trace information. The powerful tracing mechanism of Blktrace allows conclusions about request properties. The raw data of Blktrace is used to extract the actual request block relevant workload parameter metrics.

Our experiment setup is shown in Figure 5. A controller machine starts and coordinates the benchmark and monitoring tools on the system under test (SUT), which is accessed using an SSH connection. After each successful workload execution the controller machine collects the raw measurement data from the SUT and extracts the workload parameters from the raw data. Finally, the results are stored in an SQLite database.

The measurement execution sequence works as follows: i) a preparation phase that performs an initial warm up, ii) a workload execution and monitoring phase, iii) a phase that

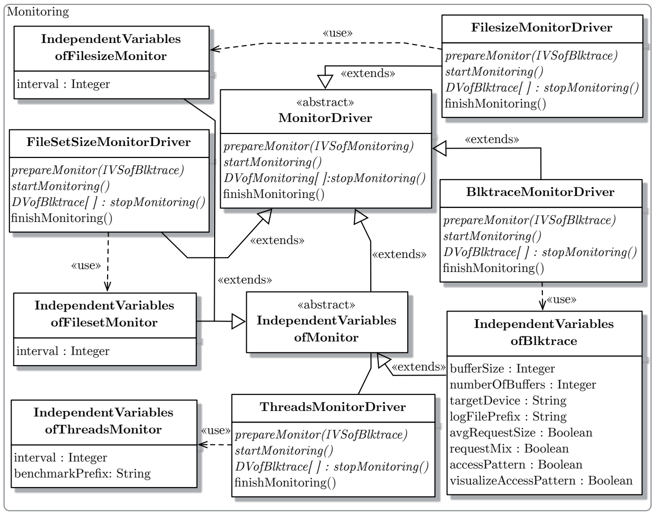


Figure 4: Class diagram of the monitor component

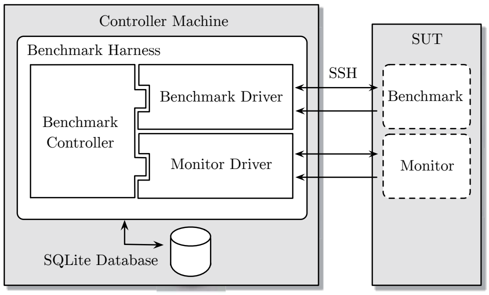


Figure 5: Experimental Setup Overview

stops monitoring and collects the measurement results from the SUT, vi) a finishing phase that finalizes the workload and monitors execution.

The parallel execution of several workloads and monitors further increases the requirements for a coordinated execution: An overlapping preparation and monitoring phase would lead to displaced characterization results. Therefore, the preparation phase must be finished before the monitoring process is started, which is automatically ensured by SPA's coordination process.

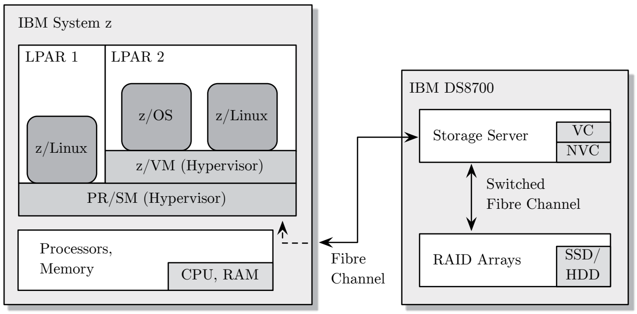


Figure 6: Conceptual design of IBM System z and DS8700

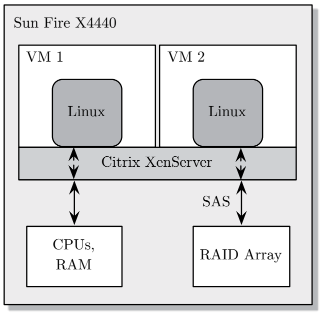


Figure 7: Conceptual design of Sun Fire X4440

3. EXPERIMENTAL SETUP

3.1 Systems Under Study

For our case studies, we use an IBM System z equipped with an IBM DS8700 storage system and a separate Sun Fire X4440 server system to evaluate how our approach performs in migration and consolidation scenarios.

For our experiments, we consider the IBM System z and the IBM DS8700 storage system as a representative virtualized environment. Both machines represent a high-end virtualized environment for critical business applications. Figure 6 shows the conceptual design of both systems.

The *Processor Resource/System Manager* (PR/SM) allows a mapping of physical to virtual resources and thus enables CPU and storage virtualization. The PR/SM hypervisor manages logical partitions (LPAR) and allocates hardware component classes (e.g. CPU, RAM, I/O) to each LPAR. An LPAR represents an independent resource container that runs applications on a Linux operating system or the z/VM hypervisor. z/VM provides an execution environment for different operating systems separated from the underlying hardware. The allocated resource class can be accessed by the LPARs either exclusively or in a shared manner. System z itself is connected to the storage system by a Fibre Channel.

The *IBM DS8700 storage system* manages I/O operations (e.g., read and write operations) using a processor complex, which contains both volatile (50 GB) and persistent (2 GB) (non-volatile) cache memory. The storage system supports a set of pre-fetching and destaging algorithms to increase the I/O performance. Write requests are cached in the volatile cache as well as in the non-volatile cache, but are destaged to the underlying RAID system asynchronously. Similar to write requests, read requests are served and cached in the volatile cache memory. Thus, recurring accesses are served by the cache, while others are served by the RAID system. The system tries to predict further reads and holds data as long as possible in its cache memory. Each processor complex can access the disk subsystem via two separate switched Fibre Channel networks (cf. [27]).

The characterization environment is set up in a z/Linux LPAR environment with 2 CPUs (more precisely, *Integrated Facilities for Linux*) and 4 GiB of memory. The storage system's partition uses the *ext4* file system. For I/O scheduling, we use a first-come, first-served (FCFS) scheduling policy.

For our migration and consolidation scenarios, we use a Sun Fire X4440 x64 server system. It contains 4 times 2.4 GhZ AMD Opteron 6 core processors and 128 GB of memory. The storage back end is a RAID system with 8 *Serial Attached SCSI* (SAS) devices with 300 GB each. It contains a write cache that allows buffering incoming write requests. The guest operating system is virtualized using a Citrix XenServer. Figure 7 shows the Sun Fire design in detail.

The consolidation scenario runs in two separate virtual machines on the Sun Fire system. Both of the virtual machines use six CPU cores and 2 GiB of memory each. The virtual machine instances access a shared RAID system. For all measurements the *ext4* file system is used. Again, for I/O scheduling, the FCFS scheduling policy is used.

3.2 Software Environment

Our software environment comprises two benchmarks and one monitoring tool: We use the Filebench benchmark to generate the original workloads to be analyzed. To this end, we enhanced SPA adding a benchmark driver to allow automated execution of Filebench.

Filebench[2] [2] is a storage system benchmark and is widely used in the performance modeling community [17, 4, 9, 19]. It supports emulation of common business workloads such as mail server and file server workloads. Its workload modeling language allows a fine-grained workload construction. Hence, Filebench achieves a realistic representation of the emulated workloads. Furthermore, we use GNU_SOURCE configuration to focus the measurements on the storage performance and to take caching effects of the storage system into account.

For validation, we use the open source FFSB benchmark. FFSB is a file system performance benchmarking tool. Our set of considered workload parameters can be mapped to FFSB's workload configuration parameters. Thus, FFSB is highly suitable to be used as a reference benchmark in our methodology. Additionally, we use the response times of FFSB as a basis to validate the characterization performance.

All benchmarking and measurement steps were executed 20 times for 300 seconds to achieve statistical robust values. Each run is prepared by a warm up period of 60 seconds.

4. CASE STUDIES

In this section, we present our case studies. In the first case study, we apply the workload characterization approach to two different workloads. In the second case study, we perform a migration scenario followed by a consolidation scenario in the context of the considered workloads.

To assess the quality of our workload characterization approach, we compare the response times of the original Filebench workload with the response time of the reference benchmark FFSB, which is supposed to emulate the original workload. As a metric in this comparison, we calculate the prediction error per run as follows. Let $RT_i \in \mathbb{R}$ be the average response times of Filebench and FFSB for one workload in run i, then,

$$RT_i^{err} = \left| \frac{RT_i^{Filebench} - RT_i^{FFSB}}{RT_i^{FFSB}} \right| \tag{15}$$

4.1 Case Study I: Workload Characterization

We show how our approach can be applied in practice and evaluate its effectiveness. We consider two different types of workloads: a mail server and a file server workload. The approach of Section 2 is realized by a four step process, illustrated in Figure 8.

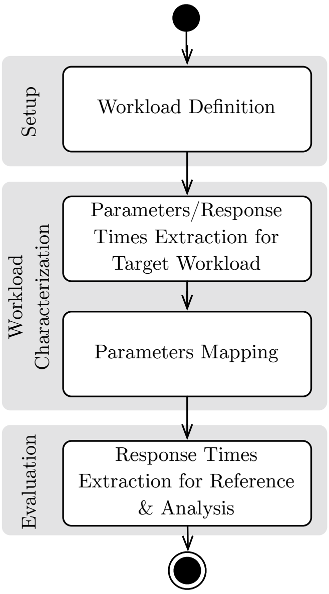

Setup
Workload Definition
Workload Characterization
Parameters/Response Times Extraction for Target Workload
Parameters Mapping
Evaluation
Response Times Extraction for Reference & Analysis

Figure 8: Execution steps in case study I

In the first step the original workload to be analyzed is specified, i.e., mail server and file server. In the second step, SPA is used to execute the original workload and collect measurements, gathering and aggregating the raw data and extracting the workload parameters. Additionally, the response times per operation are stored to allow validation of the results.
In step 3, we map the determined workload parameters to our reference benchmark. Finally, in step 4, SPA runs the reference benchmark workload configuration and extracts the response times per operation and we compare them to the response times of the original workload to assess the workload model accuracy.

Workload Definition.

Mail server and file server are basically mixed read/write workloads. They comprise a sequence of composite read and write operations. The following list gives an overview of the used operations:

- *readwholefile/writewholefile:* Both operations process one randomly chosen file as a whole, i.e., the complete file is accessed at once.
- *appendfilerand: Appendfilerand* appends a certain amount of data to the end of a file chosen randomly from the target directory.
- *openfile: Openfile* opens a file for read or write access. It supports different options to allow a random selection of a file from the target directory.
- *closefile: Closefile* simply closes a file.
- *createfile: Createfile* creates a new file in the target directory. The initial file size is 0 B.
- *deletefile:* The *deletefile* operation deletes a randomly chosen file in the target directory.
- *statfile:* The *statfile* operation accesses to the meta information of a file.

[2] https://github.com/Filebench-Revise/Filebench-Revise (bugfixed version of [2])

	Mail server	Std. dev.	File server	Std. dev.
File Size	16.62 KiB	0.74 KiB	129.76 KiB	3.91 KiB
File Set Size	683.68 MiB	58.37 MiB	1163.49 MiB	5.91 MiB
Workload Intensity	16 Threads	0 Threads	50 Threads	0 Threads
Request Size Read	14 151.17 B	31.63 B	105 152.00 B	166.67 B
Request Size Write	15 639.04 B	81.59 B	80 473.09 B	190.77 B
Request Mix	55.96 %	0.11 %	41.85 %	0.37 %
Access Pattern Ratio (Read)	28.74 %	0.52 %	97.00 %	2.30 %
Access Pattern Ratio (Write)	57.24 %	1.22 %	99.23 %	0.27 %

Table 1: Obtained metrics and standard deviations for Filebench mail and file server workload

Listing 1: Mail Server Workload

```
File set:
  - number of files = 50000
  - mean file size = 16 KiB
  - file preallocation = 80%
Threads:
  - 16 (default)
Operations:
  - deletefile
  - createfile
  - appendfilerand, mean req size = 16
    KiB
  - closefile
  - openfile,
  - readwholefile
  - closefile
  - openfile
  - appendfilerand, mean req size = 16
    KiB
  - closefile
  - openfile
  - readwholefile
  - closefile
```

Listing 2: File Server Workload

```
File set:
  - number of files = 10000
  - mean file size = 128 KiB
  - file preallocation = 80%
Threads:
  - 50 (default)
Operations:
  - createfile
  - writewholefile
  - closefile
  - openfile
  - appendfilerand, mean req size = 16
    KiB
  - closefile
  - openfile
  - readwholefile
  - closefile
  - deletefile
  - statfile
```

	Mail server	File server
File Size	17.0 KiB	130.0 KiB
Number of Files	41 201	9 169
Running Threads	16 Threads	50 Threads
Block Size Read	14 336 B	104 960 B
Block Size Write	15 872 B	80 384 B
Read Size	14 336 B	104 960 B
Write Size	15 872 B	80 384 B
Read Weight	56 %	42 %
Write Weight	44 %	58 %
Access Pt. Read	random	sequential
Access Pt. Write	sequential	sequential

Table 2: FFSB configurations for emulated mail server and file server workload

Different sequences of these operations are possible. These define the workload that the thread instance executes in a loop until the run time limit is reached. The *mail server* and *file server* operation sequences are shown in Listings 1 and 2, respectively.

Workload Parameters Extraction.

In this section, we extract the parameters of the application's workloads. Our workload characterization provides stable results all over our metrics set. Table 1 shows a comparison between the different metrics and the standard deviations of both of the considered workload scenarios. The file set size, file size, workload intensity, request mix and access pattern metrics exhibit low standard deviations. The request sizes exhibit low standard deviations in the dimensions of the actual measurement values. Figures 9, and 10 show the workload model parameter results graphically.

Workload Parameters Mapping.

The beforehand obtained workload parameters of both workloads are used to create corresponding configurations for the FFSB benchmark. As FFSB does not directly offer our set of workload metrics, we need to map our metrics to the FFSB configuration parameters. We round the request size parameters to a multiple of 512 Bytes. Table 2 shows the final FFSB parameters.

Response Time Extraction.

In this section, we show the extracted Filebench and FFSB response times for both the mail server and file server workloads. For the mail server workload, we obtain a mean response time for the read requests of 0.98 ms and 0.83 ms for the write requests. Here, the read and write response times each are comprised of two operations. Figure 11 shows the Filebench response times in detail.

For the file server workload, we observe a mean application layer response time of 11.21 ms for the read requests and

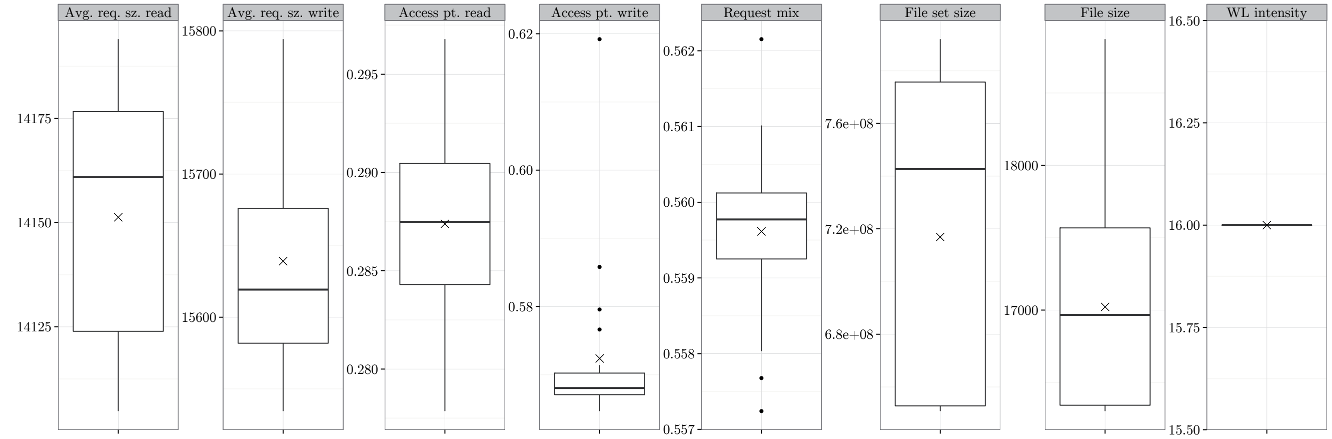


Figure 9: Filebench mail server workload parameters (units cf. Table 1)

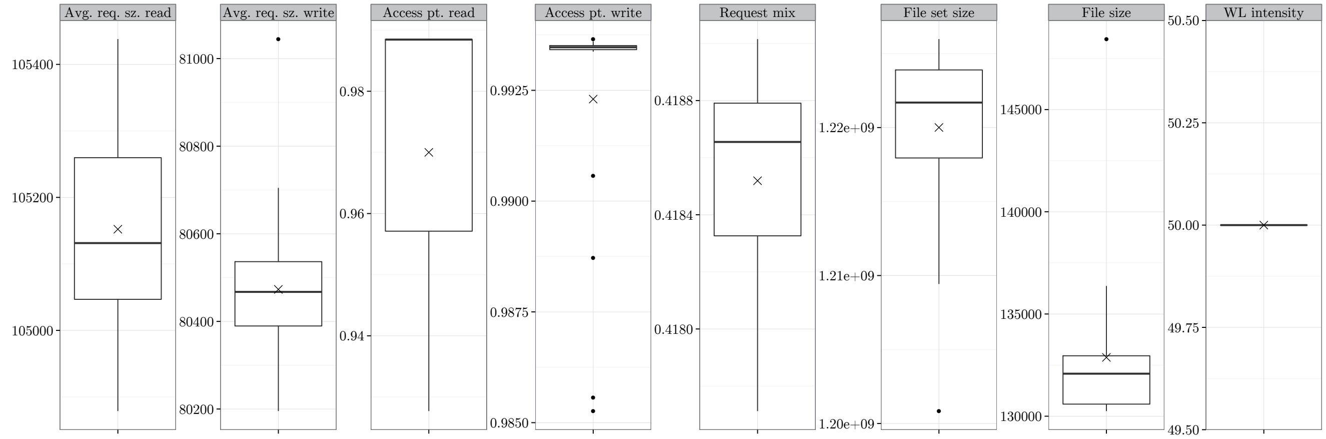


Figure 10: Filebench file server workload parameters (units cf. Table 1). The box marks the 25- to 75-percentile, the fat line is the median, the cross is the mean value, and whiskers mark the lowest (resp. highest) value still within 1.5 times the interquartile range of the lower (resp. higher) quartile.

11.65 ms for the write requests. Here, the write response time is comprised of two operations. Emulating the mail server configuration by FFSB, we obtain a read response time of 0.81 ms, while the mean standard deviation ($\overline{\sigma}$) is 2.78 ms[3]. The write response time of the mail server workload is on average 1.29 ms ($\overline{\sigma} = 2.94$ ms).

The emulated file server configuration exhibits a mean read response time of 11.67 ms ($\overline{\sigma} = 21.13$ ms). The mean write response time of the mail server workload is 18.48 ms ($\overline{\sigma} = 20.77$ ms). The response time's standard deviations in both workload scenarios result in about double the height of the actual response time values. Compare Figure 12 for the detailed FFSB response times.

Response Time Analysis.

To evaluate the accuracy of our characterization approach, we compare the response times of Filebench with the response times of the corresponding FFSB workload configuration. Figure 13 shows the obtained prediction errors.

Mail server: In the case of the mail server workload, the mean read prediction error per run is in the range of $[17.01, 25.87]\,\%$. For writes, the error is in the range of $[31.25, 38.54]\,\%$. Overall, the mean prediction error over all runs is 20.82 % for read requests and 35.72 % for write requests. Considering the very low absolute response times of this workload, these are stable results.

File server: For the file server workload's read requests response times, we observe errors in the range of $[1.18, 9.62]\,\%$. The write request errors are in the range of $[33.89, 41.06]\,\%$. Overall, the mean prediction error for read requests is 3.93 %, while the mean prediction error for write requests is 36.96 %.

Summary.

Our first case study demonstrates the performance of our workload characterization approach and its ability to capture the performance-relevant aspects of the analyzed original workloads. The slightly higher prediction error for write operations is caused by a higher standard deviation of the characterized request sizes. Our reference benchmark FFSB

[3] $\overline{\sigma}$ averages the standard deviations of each of the 20 runs.

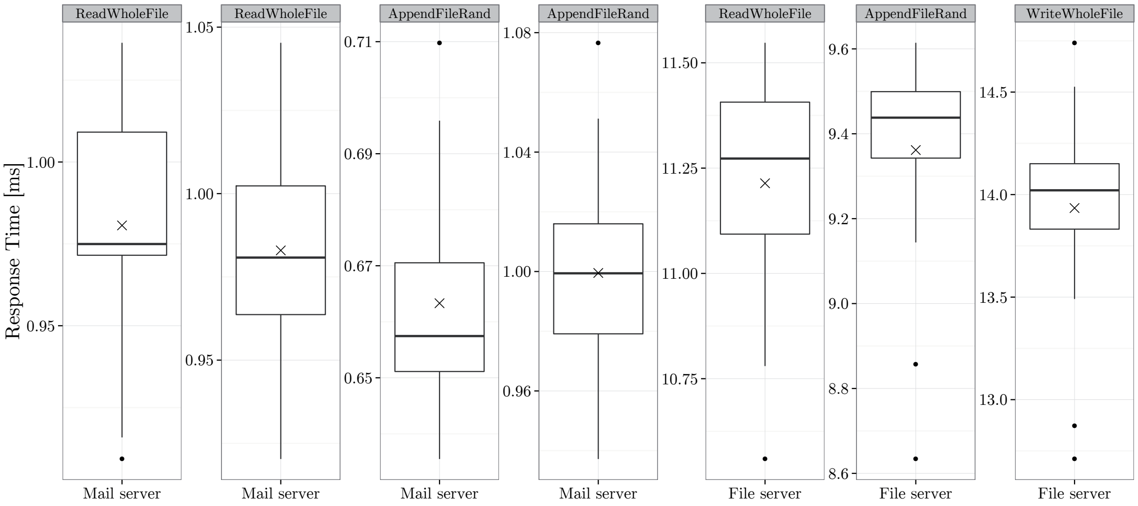


Figure 11: Filebench mail server and file server response times

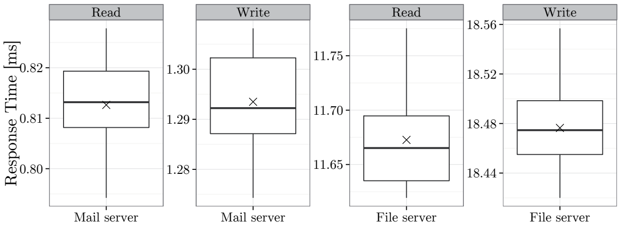


Figure 12: FFSB mail server and file server response times

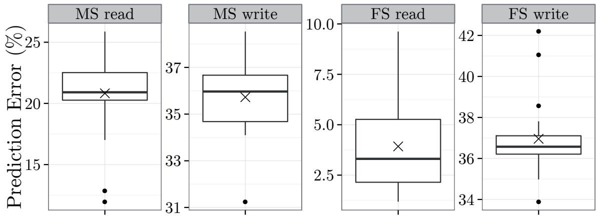


Figure 13: Filebench mail and file server prediction error

emulates constant request sizes, and therefore we use mean sizes of the requests as input. Although the prediction errors for write requests are slightly higher, the errors stay in the same dimension. Overall, the workload model accuracy is sufficient for most applications and provides an adequate characterization of the analyzed workload behavior.

4.2 Case Study II: Migration & Consolidation

In Section 4.1, we extracted the workload parameters of a mail server and a file server workload. In this case study, we use the obtained parameters to predict the performance impact of migrating and consolidating the workloads to a Sun Fire server system.

Again, we use Filebench as our original workload generator and FFSB as reference benchmark. Figures 14 and 15 illustrate the considered scenarios.

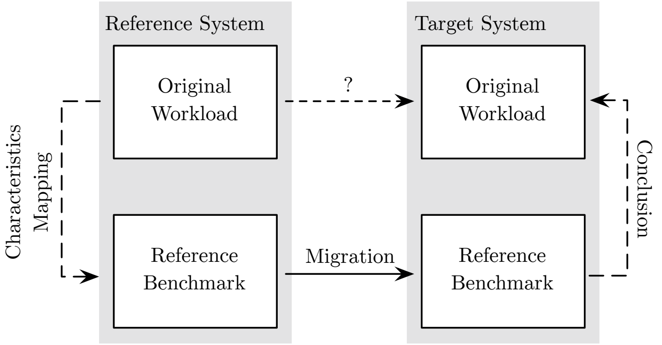


Figure 14: Migration scenario procedure

Workload Migration.

In our workload migration scenario, we show how to move a certain workload to another system. We model the workload as before and map the workload parameters to the reference benchmark. Then, we execute the migration step: We emulate the modeled workload on the target system using the reference benchmark, which allows us to evaluate the impact of the migration on the application response times. Figure 14 illustrates the procedure.

In this scenario, we use the IBM System z as reference system and the Sun Fire as target system. Like before, we use Filebench as workload generator and FFSB as reference benchmark. We use the obtained workload parameters that are described in Section 4.1. In this scenario, we concentrate on the file server workload. For evaluation, we additionally measure the Filebench response times of the file server workload on the Sun Fire system, so that we can calculate the prediction error for the workload response times.

Workload Consolidation.

The goal of the consolidation scenario is to predict the performance of two different workloads that are executed on the same machine at the same time based on workload models obtained in isolation. To do so, we characterize the workloads separately and map the obtained workload parameters to the reference benchmark. Then, both reference benchmark

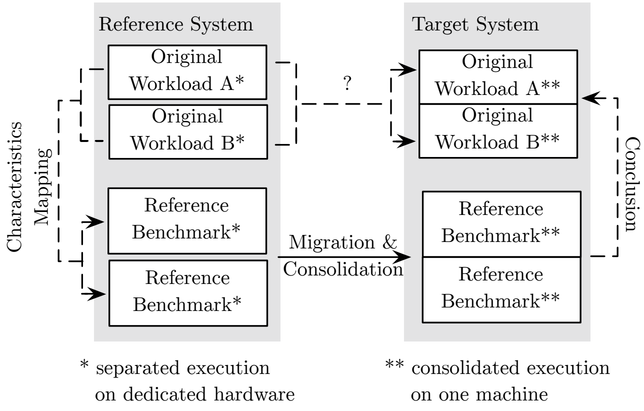


Figure 15: Consolidation scenario procedure

Workload	Operation	Resp. Time	Std. dev.
File server	ReadWholeFile	67.21 ms	0.55 ms
	AppendFileRand	34.96 ms	0.44 ms
	WriteWholeFile	31.68 ms	0.47 ms
FS + MS	AppendFileRand	85.00 ms 30.57 ms 29.94 ms	2.08 ms 0.87 ms 0.69 ms
	WriteWholeFile	93.43 ms	2.46 ms
	ReadWholeFile	139.48 ms 47.67 ms 46.11 ms	2.56 ms 1.22 ms 1.20 ms

Table 3: Details of Filebench operations response times averaged the 20 runs. The standard deviation refers to the 20 mean response times of the 20 runs.

instances are migrated to the target systems and are run consolidated, i.e., at the same time, but in separated VMs. Figure 15 illustrates this scenario.

In this consolidation scenario, again, we use the IBM System z as reference system and the Sun Fire as target system. Here, we use the workload model to predict the performance behavior when consolidating the two workloads at the same time on the Sun Fire system. Again, we use the obtained mail server and file server workload parameters of Section 4.1. For the FFSB configuration, we use the same emulated mail server and file server configurations as in Section 4.1, cf. Table 1. For the evaluation, we again extract the Filebench response times of the mail server and file server consolidated workloads on the Sun Fire system. Again, we calculate the prediction error of the workload response times.

Response Times.

In our migration scenario, for the Filebench file server workload, we obtain a mean response time of 67.21 ms, while the standard deviation (σ) is 0.55 ms for the read and 33.32 ms ($\sigma = 0.45$ ms) for the write requests. Here, the write mean response time is comprised of two operations.

For the consolidated scenario, we obtain a mean response time of 77.75 ms ($\sigma = 0.95$ ms) for the read and 59.71 ms ($\sigma = 1.02$ ms) for the write requests. Here, the read response time is comprised of three, while the write response time is comprised of four operations. For response time details of particular operations, cf. Table 3.

The FFSB mean response times for the file server workload is 55.29 ms ($\overline{\sigma} = 20.61$ ms) for the read and 42.21 ms ($\overline{\sigma} = 14.49$ ms) for the write operations. For the consolidated scenario, we obtain 89.34 ms for the read and 79.12 ms for the write operations. Figures 16(a), and 16(b) show the Filebench and FFSB scenario response times in detail.

Response Time Analysis.

As in Section 4.1, we use Equation (15) to calculate the prediction error.

Migration scenario: The prediction error of the file server workload that we used for the migration scenario shows a read error in the range of $[19.12, 28.02]$ %, and a write error in the range of $[12.77, 24.12]$ %, respectively. Overall, we obtain a mean prediction error of 21.59 % for read operations and 20.98 % for write operations.

Consolidation scenario: For the consolidation scenario, we obtain a read error of $[9.23, 17.32]$ %, and a write error of $[21.46, 28.28]$ %. Overall, we obtain a mean prediction read error of 12.95 %, while the mean write error is 24.51 %. Figure 17 illustrates the prediction error for both scenarios.

Summary.

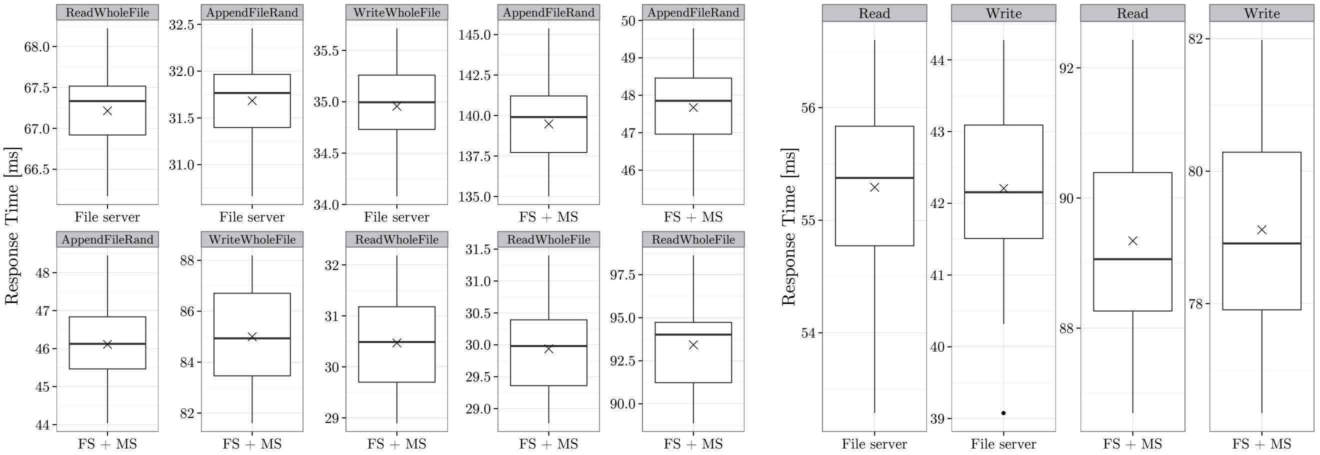
Both scenarios demonstrate the performance of our automated workload characterization approach when migrating and consolidating workloads. Our experiments cover low response times less than 30 ms, and comparatively long response times of more than 140 ms. Still, we obtain stable measurements and low prediction errors of less than 25 % for both of the considered scenarios.

5. RELATED WORK

The related work presented in this paper can be classified into three groups. The first group focuses on the area of workload characterization. Here, Gulati et al. [13], performed a study about the effects of running several workloads (i.e., a set of top-tier enterprise applications) on a shared set of I/O devices. They used a VMWare ESX server hypervisor for virtualization. They analyzed the effects of shared I/O devices while observing the behavior of workloads in isolation and in a consolidated scenario.

For their isolated workload characterization they used isolated RAID systems for each of the workloads. For their consolidated characterization they merged the isolated RAID systems into one big array. In contrast to our work, the set of workload metrics they used is more hardware related. Their analysis and conclusions concentrate on the impact of consolidating workloads with different workload access patterns. Further, they concentrate on consolidation of hard drives, but not on applications.

Ahmad et al. [5] studied the performance of storage subsystems. Here, they modeled native as well as virtualized machines. For their measurements they used several disk microbenchmarks (e.g., Iometer, Perfmon). For their characterization they concentrated on the throughput of the storage system for different block sizes, access patterns and read-/write ratios. Using the throughput of the applications they compared the native vs. virtualized performance. In a first case study, they used a capture-replay mechanism to model a



(a) Filebench scenario response times

(b) FFSB scenario response times

Figure 16: Scenarios: Migration and consolidation scenario response times

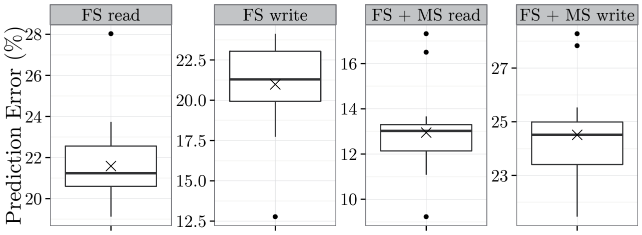


Figure 17: Scenarios prediction error

file server workload. In a second case study, they modeled a commercial mail server system. In a later work [4], they used online histograms to model several Filebench workloads. For their characterization approach they used a metrics set comprising I/O, CPU and workload intensity metrics.

Kavalanekar et al. [15] modeled several workloads, e.g., mail server and file server workloads, using the *Event Tracing for Windows* (ETW) instrumentation. They showed a metrics set for their characterization and use histograms and standardized distributions for the metric's representation. Their approach is tailored for the ETW instrumentation and uses long-term traces as input data.

Wang et al. [30] used a machine learning model CART (Classification And Regression Tree) that uses a binary decision tree to predict the storage device performance for a particular workload on a particular device. Their first approach used parameters of each disk request, i.e., its request description like arrival time differences between two requests or distances between two consecutive block accesses to predict the request/response time. The second approach used workload-level parameters like burstiness, read/write ratio, locality metrics, arrival rate and request size.

Finally, Tarasov et al. [26] extracted a workload model using I/O traces. They use their model to transfer the workload, captured by the traces, to a benchmark, i.e. Filebench. Finally, they replayed the captured workload using this benchmark.

All works in this group do not consider predicting the workload behavior on a different system or in consolidation scenarios.

The second group describes performance-influencing factors and performance metrics. Wang et at. [29] developed a statistical model to capture burstiness and spatio-temporal correlation of disk and memory accesses. They used entropy plots to represent burstiness and spatio-temporal correlation.

Ruemmler and Wilkes [25] analyze disk access patterns using disk request traces. In contrast to our work, Ruemmler and Wilkes do not respect sequential requests that are offset in time.

The third group concentrates on performance model generation. Here, Kraft et al. [18, 17] analyzed the influences of consolidated virtual environments on the response time behavior of storage systems. They used two approaches to model the I/O performance of consolidated virtual machines: First, they used homogeneous workloads in two virtual machines. In a second approach, they performed measurements for a consolidation of heterogeneous workloads. To do so, they used the FFSB benchmark as well as the Filebench benchmark. They rely on low level response times obtained by Blktrace.

Benevenuto et al. [8] developed several performance prediction models. They used open queueing models to predict the performance of applications in a XEN virtualized environment. For their model creation process they collected metrics for performance prediction model design and validation.

In contrast to the above, our workload characterization approach is a formalized and automated methodology that allows a characterization of I/O-intensive workloads using relevant parameters and enables a performance behaviour prediction on a different system and in consolidation scenarios.

6. CONCLUSIONS

We presented a fully-automated workload characterization approach that allows a systematic derivation of a workload model by capturing the major performance-relevant workload parameters. We presented our experimental methodology and the workload modeling process. We demonstrated the approach's quality on an IBM System z mainframe system equipped with a DS8700 storage system using two different workloads. On average, we achieved a prediction error of 12.38 % for the read requests and 36.34 % for the write requests.

Additionally, we studied migration and consolidation scenarios using a Sun Fire server system. The migration scenario demonstrated the ability of migrating an already modeled workload to a different system. The consolidation scenario showed the performance of our approach in virtualized environments, when moving two workloads modeled in isolation to a shared system. The migration scenario exhibits an adequate prediction error of 21.59 % for read and 20.98 % for write requests, respectively. Consolidating two prior isolated workloads results in adequate prediction errors of 12.95 % for read requests, and 24.51 % for write requests. Overall, we showed the practical relevance of our workload characterization approach, as well as its capabilities in migration and consolidation scenarios. Even though the overall prediction errors may be higher than with applying standard performance modeling techniques, the error rates are sufficient for a low-overhead, first estimation of the workload performance behavior without applying full-blown modeling formalisms or physically migrating the whole software stack.

In general, our approach is designed as a lightweight technique to evaluate the workload performance impact of an I/O-intensive software application on different platforms without requiring to actually install the whole software application. It is especially beneficial in cases that prohibit creating explicit performance models due to, e.g., time and budget constraints. As a further application scenario, our workload characterization can be used as a basis for performance modeling approaches. Existing applications can be characterized automatically and mapped to the input of the performance models, thus increasing the applicability of the performance models as well as eliminating the need for specialized performance models for a given software application.

7. ACKNOWLEDGMENTS

This work was partially funded by the German Research Foundation (DFG) under grant No. RE 1674/5-1 and KO 3445/6-1. We thank the Informatics Innovation Center (IIC) - http://www.iic.kit.edu/ - for providing the technical systems IBM System z and IBM DS8700 storage system.

8. REFERENCES

[1] FFSB. http://sourceforge.net/projects/ffsb/, 2004.

[2] Filebench. http://sourceforge.net/apps/mediawiki/filebench/, 2011.

[3] Storage performance analyzer. http://sdqweb.ipd.kit.edu/wiki/SPA/, 2013.

[4] I. Ahmad. Easy and efficient disk i/o workload characterization in vmware esx server. IEEE Computer Society, 2007.

[5] I. Ahmad, J. M. Anderson, A. M. Holler, R. Kambo, and V. Makhija. An analysis of disk performance in VMware ESX Server virtual machines. IEEE Computer Society, 2003.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. ACM, 2010.

[7] J. Axboe, A. Brunelle, and N. Scott. blktrace - linux man page. http://linux.die.net/man/8/blktrace, 2006.

[8] F. Benevenuto, C. Fernandes, M. Santos, V. A. F. Almeida, J. M. Almeida, G. J. Janakiraman, and J. R. Santos. Performance models for virtualized applications. Springer, 2006.

[9] G. Casale, S. Kraft, and D. Krishnamurthy. A model of storage i/o performance interference in virtualized systems. ICDCSW, 2011.

[10] R. C. Chiang and H. H. Huang. Tracon: interference-aware scheduling for data-intensive applications in virtualized environments. New York, NY, USA, 2011. SC'11.

[11] J. Gantz and D. Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. IDC, 2012.

[12] B. Gregg. Dtrace tools: iopattern. http://www.dtracebook.com/index.php/Disk_IO:iopattern, Jul-2005. [Online; accessed 21-Aug-2013].

[13] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. VPACT, 2009.

[14] V. Kasavajhala. Solid State Drive vs. Hard Disk Drive Price and Performance Study. Technical report, Dell Technical White Paper. Dell PowerVault Storage Systems, 2011.

[15] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda. Characterization of storage workload traces from production windows servers. IISWC, 2008.

[16] S. Kounev. Wiley encyclopedia of computer science and engineering - chapter software performance evaluation. Wiley-Interscience and John Wiley & Sons Inc., 2009.

[17] S. Kraft. Performance models of storage contention in cloud environments. *Journal of Software and Systems Modeling (SoSyM)*, 2012.

[18] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick. IO performance prediction in consolidated virtualized environments. ICPE, 2011.

[19] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta. Modeling virtualized applications using machine learning techniques. SIGPLAN/SIGOPS, 2012.

[20] I. R. Limited. Global server virtualization market 2012-2016. http://www.rnrmarketresearch.com/global-server-virtualization-market-2012-2016-market-report.html, 2013. [Online; accessed 19-Aug-2013].

[21] L. McLaughlin. Virtualization in the enterprise survey: Your virtualized state in 2008. *CIO Magazine*, 2008.

[22] Q. Noorshams, D. Bruhn, S. Kounev, and R. Reussner. Predictive Performance Modeling of Virtualized Storage Systems using Optimized Statistical Regression Techniques. ICPE '13, 2013.

[23] Q. Noorshams, S. Kounev, and R. Reussner. Experimental Evaluation of the Performance-Influencing Factors of Virtualized Storage Systems. Springer Berlin Heidelberg, 2013.

[24] Q. Noorshams, K. Rostami, S. Kounev, P. Tůma, and R. Reussner. I/O Performance Modeling of Virtualized Storage Systems. MASCOTS, 2013.

[25] C. Ruemmler and J. Wilkes. Unix disk access patterns. Hewlett-Packard Laboratories, 1993.

[26] Tarasov, Kumar, Ma, Hildebrand, Povzner, Kuenning, and Zadok. Extracting flexible, replayable models from large block traces. FAST'12, 2012.

[27] R. Vaupel. *High Availability and Scalability of Mainframe Environments using System z and z/OS as example*. 2012.

[28] W. Vogels. Beyond server consolidation. ACM, 2008.

[29] M. Wang, A. Ailamaki, and C. Faloutsos. Capturing the spatio-temporal behavior of real traffic data. Elsevier Science Publishers B. V., 2002.

[30] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. Ganger. Storage device performance prediction with cart models. MASCOTS, 2004.