Experimental Evaluation of the Performance-Influencing Factors of Virtualized Storage Systems

Qais Noorshams, Samuel Kounev, and Ralf Reussner

Chair for Software Design and Quality
Karlsruhe Institute of Technology (KIT)
Am Fasanengarten 5, 76131 Karlsruhe, Germany
{noorshams,kounev,reussner}@kit.edu

Abstract. Virtualized cloud environments introduce an additional abstraction layer on top of physical resources enabling their collective use by multiple systems to increase resource efficiency. In I/O-intensive applications, however, the virtualized storage of such shared environments can quickly become a bottleneck and lead to performance and scalability issues. In software performance engineering, application performance is analyzed to assess the non-functional properties taking into account the many performance-influencing factors. In current practice, however, virtualized storage is either modeled as a black-box or tackled with full-blown and fine-granular simulations. This paper presents a systematic performance analysis approach of I/O-intensive applications in virtualized environments. First, we systematically identify storage-performance-influencing factors in a representative storage environment. Second, we quantify them using a systematic experimental analysis. Finally, we extract simple performance analysis models based on regression techniques. Our approach is applied in a real world environment using the state-of-the-art virtualization technology of the IBM System z and IBM DS8700.

Keywords: I/O, Storage, Performance, Virtualization.

1 Introduction

Today's growth in resource requirements demand for a powerful yet versatile and cost-efficient data center landscape. Virtualization is used as the key technology to cost-optimally increase resource efficiency, resource demand flexibility, and centralized administration. Furthermore, cloud environments enable new pay-per-use cost models to provide resources on-demand.

Modern cloud applications have increasingly an I/O-intensive workload profile (cf. [1]), e.g., mail or file server applications are often deployed in virtualized environments. With the rise in I/O-intensive applications, however, the virtualized storage of shared environments can quickly become a bottleneck and lead to unforeseen performance and scalability issues.

In current software performance engineering approaches, however, virtualized storage and its performance-influencing factors are often neglected. The virtualized storage is either treated as a black-box due to its complexity or modeled with full-blown and fine-granular simulations. In rare cases, elaborate analysis is applied, e.g., [2,3], however without explicitly considering storage configuration aspects and their influences on the storage performance.

This paper presents a systematic performance analysis approach for I/O-intensive applications in virtualized environments. First, we systematically identify storage-performance-influencing factors by considering a representative virtualized storage environment. These influencing factors are modeled hierarchically and organized categorically by workload, operating system, and hardware. The studied influencing factors and their classification are of general nature and not specific to the considered environment. Second, we propose a general workload and benchmarking methodology in order to reason about the performance of I/O-intensive applications in virtualized environments. By applying our methodology, we quantify the influence of the identified factors by means of a systematic experimental analysis. Finally, we derive simple performance analysis models based on linear regression that can be used in software performance engineering. The approach is applied in a real world environment using the state-of-the-art virtualization technology of the *IBM System z* and *IBM DS8700* with full control of the system environment and the system workload.

In summary, the contributions of this paper are: i) We provide a study of virtualized storage performance in a real world environment using the state-of-the-art virtualization technology of the IBM System z. ii) We systematically identify storage-performance-influencing factors and abstract them by means of a hierarchical feature tree model. iii) We provide an in-depth quantitative evaluation and comparison of the storage-performance-influencing factors. iv) We create simple regression-based performance models for performance prediction of a variety of storage workloads.

The remainder of this paper is organized as follows. Section 2 presents our system environment. Section 3 identifies storage-performance-influencing factors. Section 4 presents our experimental methodology and evaluation. In Section 5, we extract simple performance models based on linear regression. Finally, Section 6 presents related work, while Section 7 summarizes.

2 System Environment

A typical virtualized environment in a data center consists of servers connected to storage systems. The mainframe *System z* and the storage system *DS8700* of IBM comprise our virtualized environment of focus. They are state-of-the-art high-performance virtualized systems with redundant and/or hot swappable resources for high availability. The System z combined with the DS8700 represent a typical cloud environment. The System z enables on-demand elasticity of pooled resources with an inherent pay-per-use accounting system (cf. [4]). The System z provides processors and memory, whereas the DS8700 provides storage space. The structure of this environment is illustrated in Figure 1.

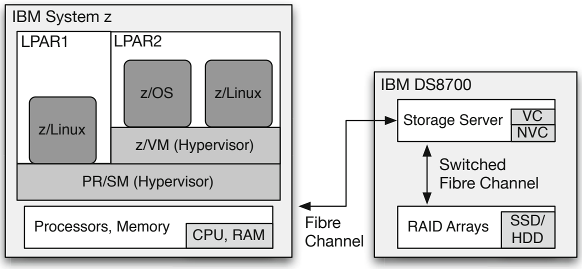


Fig. 1. IBM System z and IBM DS8700

The *Processor Resource/System Manager (PR/SM)* is a hypervisor managing logical partitions (*LPARs*) of the machine (therefore also called *LPAR hypervisor*) and enabling CPU and storage virtualization. For memory virtualization and administration purposes, IBM introduces another hypervisor, *z/VM*. The System z supports the classical mainframe operating system *z/OS* and special Linux ports for System z commonly denoted as *z/Linux*. The System z is connected to the DS8700 via fibre channel. Storage requests are handled by a storage server having a volatile cache (VC) and a non-volatile cache (NVC). Write-requests are written to the volatile as well as the non-volatile cache, but they are destaged to disk asynchronously. The storage server is connected via switched fibre channel with SSD and/or HDD RAID arrays.

In such multi-tiered virtualized environments with several layers between the hosted applications and the physical storage, many questions arise concerning the impact of the workload profile and the storage configuration on the system performance. How do certain workload characteristics, e.g., read/write request ratio, affect the performance? How does the locality of requests affect performance in tiered environments? Further, do current standard system configurations, e.g., the default I/O scheduler, still show to be suitable in such an environment? Our experiments in Section 4 will examine these and more questions.

3 Storage Performance Influencing Factors

For the identification of storage-performance-influencing factors, we created a hierarchical feature tree model aligned along logical borders, i.e., between workload mix and system configurations, and system borders, i.e., between System z and DS8700. The models capturing the performance-influencing factors are shown in Figure 2, however, complex interactions between factors are not depicted for the sake of simplicity. Still, workload-specific effects on system configurations are discussed in our experiment scenarios in Section 4.

3.1 Workload Mix

The top model in Figure 2 shows basic characteristics of a workload mix. The characterization is based on a general, storage-level view on the workload.

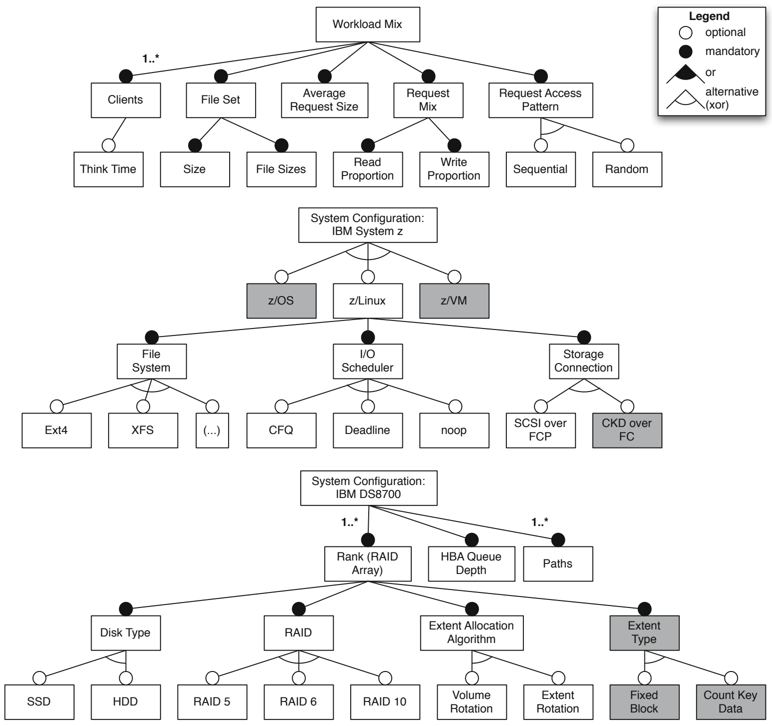


Fig. 2. Storage Performance Influencing Factors (in gray: system-specific factors)

- *Clients*: The requests of the workload are created by a certain number of clients representing e.g., threads or processes. After completion of a request, clients may have a certain waiting time (or *think time*), e.g., to process the data requested, before issuing another request. The number of clients affects performance defining the number of concurrent requests that require scheduling and introduce resource contention.
- *File Set*: Applications read from and write into a set of files. The size of this *file set* influences the locality of requests and the effectiveness of caching algorithms and data placement strategies. Depending on the physical allocation of the files, the *file sizes* affect, i.e., limit, sequential requests.
- *Average Request Size*: Most I/O optimization strategies in the various layers of the storage system aim at maximizing throughput by merging subsequent sequential requests. Serving many small requests results in a lower throughput than serving fewer large requests.
- *Request Mix*: While *read* requests are synchronous, *write* requests can be served asynchronously without blocking the application. This leads to complex optimization strategies when having mixed requests.
- *Request Access Pattern*: The access pattern affects performance due to the physical access of data as well as the optimization strategies in the various layers of

the storage system. Typical request access pattern are random or sequential requests.

3.2 System Configuration

The system configuration space is separated into two systems, System z and DS8700. While the former represents the computing environment including operating system configurations, the latter represents the storage environment including hardware configurations, cf. the middle and bottom models in Figure 2, respectively.

IBM System z – Computing Environment

- *IBM System z*: As mentioned before, the System z runs different operating systems, *z/OS* and *z/Linux*, which both can run on another virtualization layer *z/VM* to ease administration and increase resource sharing.
- *z/Linux*: As previously described, *z/Linux* is a special Linux port for System z and can be regarded as a regular Linux system.
- *File System*: Modern *file systems*, e.g., *EXT4* as the de facto standard for Linux or *XFS*, exhibit significant performance differences under the same workload as they are implemented and optimized differently.
- *I/O Scheduler*[1]: The current Linux standard *completely fair queueing (CFQ)* scheduler performs several optimizations (e.g., splitting/merging and request reordering) to minimize disk seek times, which account for a major part of I/O service times in disk-based storage systems. The *deadline* scheduler imposes a deadline on requests to prevent request starvation with read requests having a significantly shorter deadline than write requests. The *no operation (noop)* scheduler only merges and splits disk requests.
- *Storage Connection*: The System z provides two different protocols embedded in a protocol for fibre channel, classical mainframe protocol (*CKD*) over fibre channel (i.e., FICON protocol) or widespread *SCSI* over fibre channel protocol (*FCP*). The required protocol depends on the storage volume format (cf. *Extent Type*).

IBM DS8700 – Storage Environment

- *IBM DS8700*: The storage system has several configuration parameters, which - although not all directly - can be mapped to configurations of different systems.
- *Rank (RAID Array)*: A RAID array formatted with a type (*extent type*) and sub-divided into equally sized partitions (*extents*) is a *rank*. The extents are used to define volumes that can be used by applications.

[1] Note: i) The *antecepatory* scheduler was removed from the latest Linux kernel and is highly discouraged in virtualized environments. ii) The recently announced *FIOPS* scheduler is designed for flash-based storage devices and is still under development.

  - *Disk Type*: An array can be created with disks of a specific *disk type* advantageous for different usages: Fast, but more expensive *SSDs* for higher performance requirements or regular *HDDs* (with either 7.2k r/min or 15k r/min) for lower cost per storage space.
  - *RAID*: The different RAID types offer a trade-off between performance, reliability and resource efficiency.
  - *Extent Type*: The format type of an array can be classical mainframe format, i.e., *Count Key Data (CKD)*, optimized for availability or regular format, i.e., *Fixed Block (FB)*. Each type requires a different protocol (cf. *Storage Connection*).
  - *Extent Allocation Algorithm*: Ranks (possibly a SSD/HDD mix) can be pooled. Using multiple ranks to create volumes, the extents of a volume can be allocated on one rank after the other or be striped across ranks to optimize performance by exploiting parallelism.
- *HBA Queue Depth*: The host bus adapter (HBA) receives and queues all requests to the storage system. Having multiple servers accessing the storage system, the HBA queue depth controls fairness among servers, e.g., if one server sends much more storage requests than another.
- *Number of Paths*: To improve availability, multiple logical or physical paths can be defined from an operating system to the storage system. However, this induces routing overhead to determine which path to use for the next request.

4 Experimental Storage Performance Analysis

In this section, we analyze the storage-performance-influencing factors presented in Section 3. We start by introducing our experimental methodology.

4.1 Experimental Methodology

The experimental environment consists of the System z connected to the DS8700 storage system as introduced in Section 2 and illustrated in Figure 1.

As load driver, we used the open source Flexible File System Benchmark (FFSB)[2] because of its fine-grained configuration possibilities needed for our in-depth analysis. The detailed parameters of the workload and the system considered in our experiments are shown in Table 1. The *fixed* parameters are not varied in the experiments. The *variable* parameters are chosen depending on the specific analysis scenario. The workload parameters marked as *full explore* are measured in all combinations in every considered scenario. In our experiments, hardware variations are out of scope due to constraints in the available hardware configuration.

As an example of a benchmark run, assuming 4KB requests and a read/write ratio of 80%/20%, 100 threads repeatedly issue a series of 4096 requests[3] for

[2] `http://sourceforge.net/projects/ffsb/`. Note: There is a known bug in FFSB that miscalculates the throughput, however, it was fixed in the version we used.

[3] This is limited by the smallest file size and the largest request size (128MB/32KB = 4096) and is kept constant in all runs to ensure comparability.

Table 1. Experiment Setup and Parameters

(a) Workload Configurations

Fixed	
Runtime	300sec
Threads	100 (no think time)
`O_DIRECT`[4]	on
Variable	
File set size	{10x 128MB, 10x 1GB, 150x 1GB}
Full explore	
Read/Write Ratio	{100%/0%, 80%/20%, 50%/50%, 20%/80%, 0%/100%}
Request size	{4KB, 32KB}
Access pattern	{random, sequential}

(b) System Configurations

Fixed	
OS	z/Linux (Debian 2.6.32-5-s390x)
CPU	2 IFLs (cores) with approx. 2760 MIPS
RAM	4GB
Storage Cache	50GB (volatile), 2GB (non-volatile)
Storage Array	One RAID5 array with 7 HDDs (15k r/min)
Connection	SCSI over FCP
Variable	
Hypervisor	{`LPAR`, `z/VM` (on top of LPAR)}
File system	{`EXT4`, `XFS`}
Scheduler	{`CFQ`, `deadline`, `noop`}

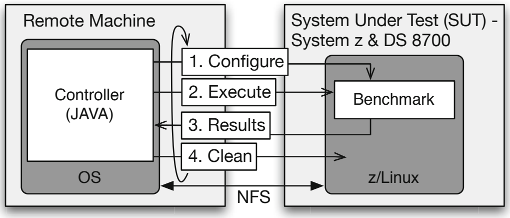


Fig. 3. Experimental Controller Setup

5 minutes. Each thread issues a request as soon as the previous request is completed. For each request series, a thread issues with 80% probability a read request series and with 20% probability a write request series. Thus, during this benchmark run, the system will be exposed to a stochastically mixed read and write workload while the benchmark gathers performance data for both types of requests. If the requests are sequential, the logical addresses of subsequent requests of one thread are ascending and 4KB apart. Further, since having, e.g., 100% read requests implies that there are no write requests during one run, such benchmark runs cannot induce any write performance data (cf. fewer bars in the figures in Section 4.2 depicting 100% read and 100% write requests respectively).

The experiments are fully automated and run by a controller software written in JAVA. Illustrated in Figure 3, the controller software is located on a remote machine that is connected to the System z via NFS. After configuring the controller, the configuration space of the experiments is explored. For every (benchmark) configuration, the controller i) configures the benchmark, ii) executes the benchmark, iii) collects the results during and after the run to the remote machine to not further introduce load on the system under test (SUT), and finally iv) cleans the system by removing the files written by the benchmark.

[4] POSIX flag that minimizes caching effects of the host.

For the analysis, we focus on the average system performance, thus, we compared the mean response time and mean throughput for each workload and configuration scenario. We repeated each benchmark run 15 times. As the response times are very small, the means calculated by the benchmark are prone to large outliers. In order to obtain stable and meaningful results, we ordered the response time means, and calculated the mean of the middle 5 values such that the results can be reproduced.

4.2 Experimental Results

In our evaluation, we first provide an analysis of the performance influences of the workload profile under a representative system setting. We then analyze performance characteristics of the system environment, specifically storage level caching effects and z/VM hypervisor overhead. Finally, we analyze operating system influences by varying the file system and the I/O scheduler.

Workload – *Variable Parameters:* `LPAR` hypervisor, `EXT4` file system, `noop` scheduler, `1280MB` file set.

We evaluate the system performance under varying read/write ratio, as well as performance of different sized and of sequential and random access requests.

Figures 4a, 4b show the performance of read and write requests depending on the read/write ratio. The metrics are normalized w.r.t the 100% read and the 100% write workload respectively. The results show that sequential read-requests have higher response times under a higher write-request fraction. This is because the read-requests are retained (i.e., queued) longer under a higher write-request fraction if the system recognizes the sequential access pattern. The goal is to serve multiple read-requests at once to improve throughput. In this scenario, the throughput of sequential and random access read-requests are approximately equal, however, the benefit of this behavior becomes evident later when examining the caching effects. In contrast to the read-requests, the response times of sequential write-requests are lower than the response time of random access write-requests under higher read-request proportion. Again, throughput of sequential and random access write-requests are approximately equal.

To elaborate, Figures 4c, 4d illustrate the relative differences in the measured response time and throughput between sequential and random access requests where the metric values for sequential requests are used as a reference, i.e., a negative value corresponds to a lower value of the respective metric for random requests. Except for the case of 100% read-requests, the sequential requests are always slower than random access requests, for read-requests between 50% and 70% and for write-requests between 20% and 50%, depending on the read/write ratio. However, comparing the throughput metric of the configuration confirms that the differences in throughput are negligible, i.e., less than 4% in most cases and up to 7% in one case.

The previous measurements show no noticeable difference between 4KB and 32KB requests. Figures 4e, 4f illustrate the relative differences of response time and throughput between 4KB (used as reference) and 32KB requests.

As expected, response time and throughput of 32KB requests are consistently higher than of 4KB requests with write- (read-) requests having higher (lower) response times in write-intensive workloads. However, this behavior is specific to this scenario and, as we will show later, it is different for the `XFS` file system.

Caching Effects – *Variable Parameters:* `LPAR` hypervisor, `EXT4` file system, `noop` scheduler, {`1280MB, 10GB, 150GB`} file set.

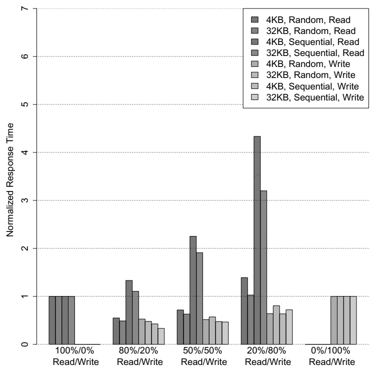
As described before, this system environment consists of two storage tiers comprising storage server cache and RAID array. These tiers have significantly different performance characteristics. The experimental results in this scenario demonstrate the need for intelligent read-ahead and de-staging algorithms in heterogeneous or multi-tier storage environments. They are illustrated in Figure 5 comparing 1280MB and 10GB as well as 1280MB and 150GB file sets. As explained in [5], the storage system combines the algorithms Sequential Adaptive Replacement Cache (SARC), Adaptive Multi-stream Prefetching (AMP), and Intelligent Write Caching (IWC). This combination was shown to be very effective for workloads exhibiting a certain predictable access pattern, e.g., sequential requests. In case of a 10GB file set, the data exceeds the non-volatile cache leading to frequent de-staging of data from the non-volatile cache to the disk array. In case of a 150GB file set, the file set also exceeds the volatile cache leading to frequent cache misses especially for random access read-requests.

z/VM Layer – *Variable Parameters:* {`LPAR,z/VM`} hypervisor, `EXT4` file system, `noop` scheduler, `1280MB` file set.

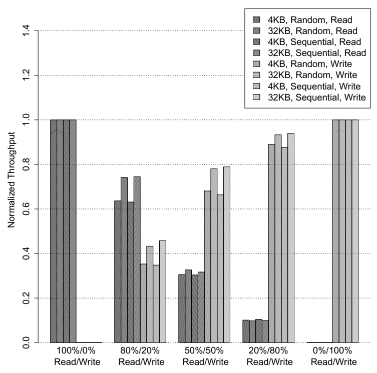
Figure 6 illustrates the relative differences in terms of observed response time and throughput when adding another hypervisor layer (using z/VM). This setup benefits larger requests and impairs smaller requests in mixed workloads in terms of the relative response times, however, given that the absolute values are less than 2ms, the respective absolute performance differences are rather low. Furthermore, the throughput differences are not significant given that a SCSI connection is used and the additional hypervisor layer only performs address mapping/translation tasks.

File System – *Variable Parameters:* `LPAR` hypervisor, {`EXT4,XFS`} file system, `noop` scheduler, `1280MB` file set.

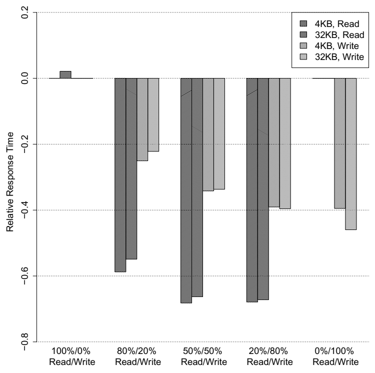
Figure 7 illustrates the relative differences in terms of observed response time and throughput between `EXT4` (used as reference) and `XFS`. This comparison requires a more fine-grained analysis as it exhibits a number of patterns. While for pure read-workloads the read performance is constant, `XFS` is able to perform better than `EXT4` for sequential reads in mixed workloads. For random reads, `XFS` exhibits higher response times for smaller read-requests. Large read-requests in a write-intensive workload (read/write = 20%/80%) is the only case where `XFS` performs better than `EXT4` for random reads. For write-requests, `XFS` shows to have higher response times than `EXT4` in mixed workloads. Furthermore, even for pure write workloads, `XFS` has about 50% higher response times for small random



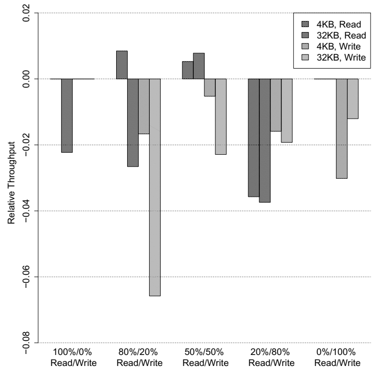
(a) Normalized RT



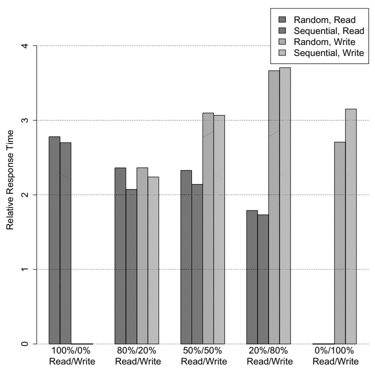
(b) Normalized TP



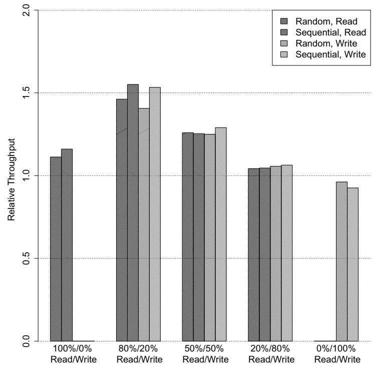
(c) Relative Sequential/Random RT Differences



(d) Relative Sequential/Random TP Differences

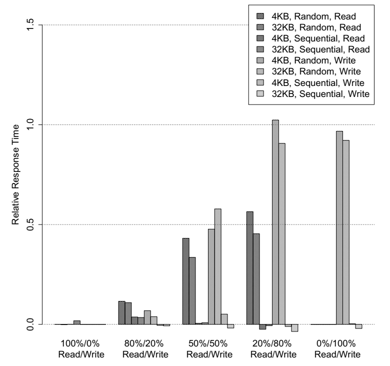


(e) Relative 4KB/32KB RT Differences

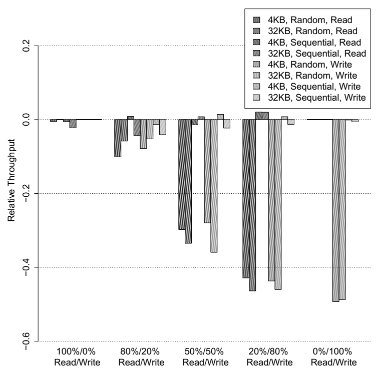


(f) Relative 4KB/32KB TP Differences

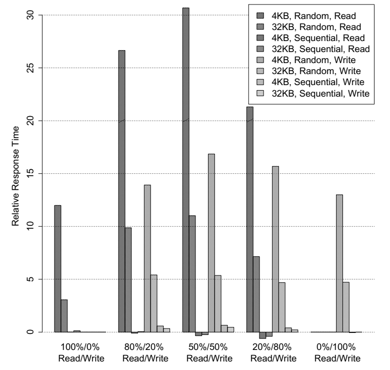
Fig. 4. Workload Performance: Response Time (RT) and Throughput (TP)



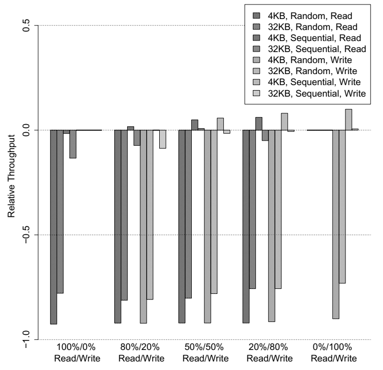
(a) Relative RT Differences between 1280MB and 10GB File Sets



(b) Relative TP Differences between 1280MB and 10GB File Sets



(c) Relative RT Differences between 1280MB and 150GB File Sets



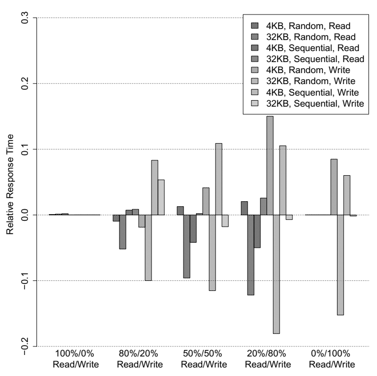
(d) Relative TP Differences between 1280MB and 150GB File Sets

Fig. 5. Caching Effects when Varying File Set Sizes

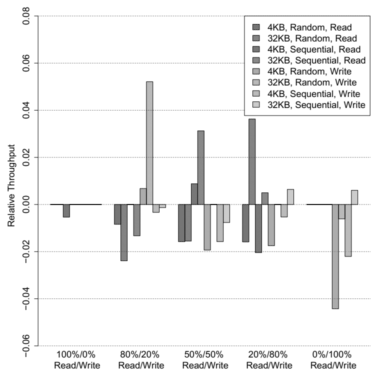
writes-requests. However, `XFS` is able to improve throughput for most workloads, most noticeably for small random requests in write-intensive workloads.

I/O Scheduler – *Variable Parameters:* `LPAR` hypervisor, `EXT4` file system, {`CFQ, deadline, noop`} scheduler, `1280MB` & `150GB` file set.

As the standard scheduler in Linux distributions, CFQ incorporates a so-called elevator mechanism reordering I/O-requests to minimize disk seek times. Figure 8 illustrates the relative differences in terms of observed response time and throughput between the `noop` (used as reference) and `CFQ` I/O schedulers, the top two charts for a file set size of 1280MB, the bottom ones for 150GB. The scheduler queues (especially small) read-requests longer if less write-requests are in the queue, trying to reorder requests and to merge small requests into larger ones. However, while this optimization is important for regular disks and disk arrays, in this tiered environment, this scheduling shows drastic performance degradation. Even if the cache is fully utilized and the response time of random

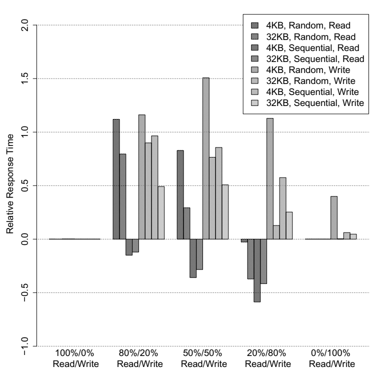


(a) RT Differences

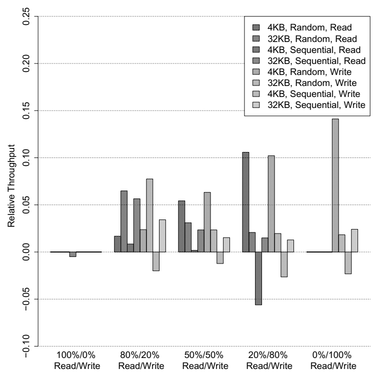


(b) TP Differences

Fig. 6. Relative Performance Overhead of z/VM hypervisor



(a) RT Differences



(b) TP Differences

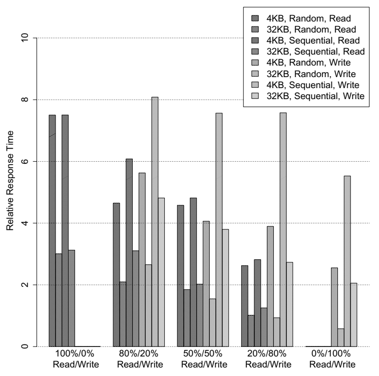
Fig. 7. Relative Performance Differences of `EXT4` and `XFS` File System

read-requests in write-intensive workloads is slightly decreased, the throughput is significantly reduced with this scheduler.

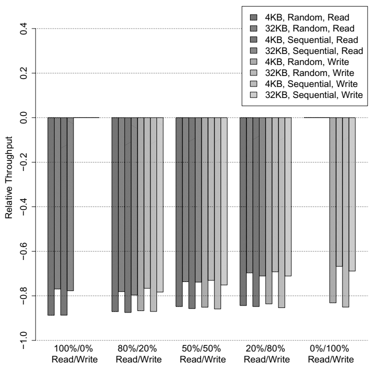
The `deadline` scheduler assigns deadlines to requests in order to avoid request starvation. Interestingly, in our environment, it exhibited almost no performance differences (less than 5%, mostly less than 1%) to the `noop` scheduler, therefore, the results are omitted for brevity.

4.3 Discussion

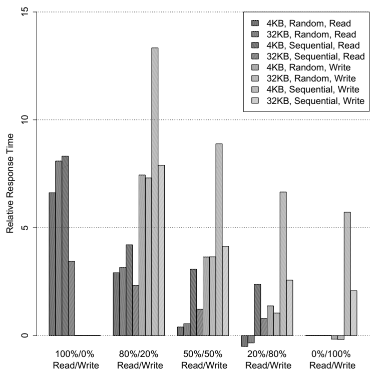
Summarizing the measurement results, the conclusions from our analysis are manifold. In our tiered storage environment, the workload is subject to optimization and queueing in the storage system. The applied read ahead mechanisms (i.e., publicly available caching algorithms) improve performance significantly for requests with



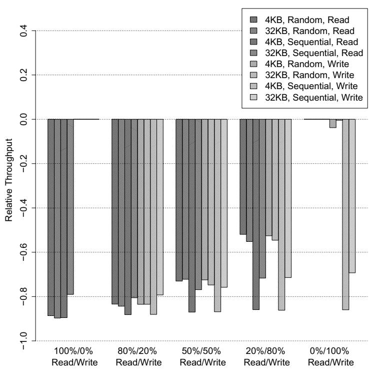
(a) RT Differences, 1280MB File Set



(b) TP Differences, 1280MB File Set



(c) RT Differences, 150GB File Set



(d) TP Differences, 150GB File Set

Fig. 8. Relative Performance Differences of `noop` and `CFQ` Schedulers

certain access pattern (e.g., sequential). However, the increased queueing of sequential read-requests impairs the response time on fully cached data considerably while still achieving equal throughput of sequential and random requests. More specifically, the response time is highly dependent on the read/write ratio. For fully cached data, sequential read-requests have considerably higher response times under a higher write-request fraction. However, the performance of sequential requests is approximately stable and independent of the file set size. Considering the hypervisor, z/VM introduces very low virtualization overhead. More generally, while the `EXT4` file system is very widespread being the de facto standard for Linux systems, the `XFS` file system showed to be beneficial for sequential reads in terms of response time. In terms of throughput, `XFS` is beneficial for big requests in read-intensive workloads and for random read- and write-requests for balanced and write-intensive workloads. Furthermore, while the `noop` and the `deadline` I/O scheduler showed to perform equally well in our environment, the `CFQ` scheduler impaired performance drastically.

Table 2. Linear Regression Depending on Type (Read or Write) Proportion

(a) Response Time (file set: 1280MB)

Size	Access	Type	R^2
4KB	Random	Read	0.29346
32KB	Random	Read	0.03769
4KB	Sequential	Read	**0.92605**
32KB	Sequential	Read	**0.93349**
4KB	Random	Write	0.69652
32KB	Random	Write	**0.9537**
4KB	Sequential	Write	0.81183
32KB	Sequential	Write	**0.94942**

(b) Throughput (file set: 1280MB)

Size	Access	Type	R^2
4KB	Random	Read	**0.96063**
32KB	Random	Read	**0.98413**
4KB	Sequential	Read	**0.95726**
32KB	Sequential	Read	**0.98101**
4KB	Random	Write	**0.9785**
32KB	Random	Write	**0.93358**
4KB	Sequential	Write	**0.98546**
32KB	Sequential	Write	**0.93377**

(c) Response Time (file set: 150GB)

Size	Access	Type	R^2
4KB	Random	Read	**0.98003**
32KB	Random	Read	**0.96287**
4KB	Sequential	Read	**0.99828**
32KB	Sequential	Read	**0.925**
4KB	Random	Write	0.89803
32KB	Random	Write	**0.94982**
4KB	Sequential	Write	**0.99879**
32KB	Sequential	Write	**0.99711**

(d) Throughput (file set: 150GB)

Size	Access	Type	R^2
4KB	Random	Read	**0.97489**
32KB	Random	Read	**0.95248**
4KB	Sequential	Read	**0.96628**
32KB	Sequential	Read	**0.99029**
4KB	Random	Write	**0.99323**
32KB	Random	Write	**0.98821**
4KB	Sequential	Write	**0.98905**
32KB	Sequential	Write	**0.93505**

5 Regression-Based Storage Performance Modeling

In order to effectively extract storage performance models, we apply linear regression using the read/write ratio as independent and the performance as dependent variables. The aim is to approximate the performance of mixed workloads. This analysis is applied once for a 1280MB and a 150GB file set respectively. The further system configurations are set to `LPAR` hypervisor, `noop` I/O scheduler, and `EXT4` file system.

For a 1280MB file set, Table 2a and Table 2b show the coefficient of determination R^2 for linear regression models of the system performance when varying the proportion of the operation type. For the response time, sequential read-requests and 32KB write-requests exhibit a strong linear fit. For the throughput, all workloads exhibit a strong fit. In these cases, the linear performance model show to be effective approximations of the real performance. For random read requests and small write requests, we reason that the request queueing and scheduling in the storage system predominate the response time if the file set is fully cached. Therefore, the linear regression models fit nicely unless considering the pure workload for these scenarios, cf. Figure 4a.

For a 150GB file set, the result in Table 2c and Table 2d show to effectively approximate response time and throughput of workloads for varying read/write ratios. Even though there is one lower R^2 value, the approximation is still acceptable.

6 Related Work

Many general modeling techniques for storage systems exist, e.g., [6,7,8], but they are only shortly mentioned here as our work is focused on virtualized environments. The work closely related to the approach presented in this paper can be classified into two groups. The first group is focused on modeling storage performance in virtualized environments. Here, Kraft et al. [2] present two approaches based on queueing theory to predict the I/O performance of consolidated virtual machines. Their first, trace-based approach simulates the consolidation of homogeneous workloads. The environment is modeled as a single queue with multiple servers having service times fitted to a Markovian Arrival Process (MAP). In their second approach, they predict storage performance in consolidation of heterogeneous workloads. They create linear estimators based on mean value analysis (MVA). Furthermore, they create a closed queueing network model, also with service times fitted to a MAP. In [9], Ahmad et al. analyze the I/O performance in VMware's ESX Server virtualization. They compare virtual to native performance using benchmarks. They further create mathematical models for I/O throughput predictions. To analyze performance interference in a virtualized environment, Koh et al. [10] manually run CPU bound and I/O bound benchmarks. While they develop mathematical models for prediction, they explicitly focus on the consolidation of different types of workloads, i.e., CPU and I/O bound. By applying an iterative machine learning technique, Kundu et al. [11] use artificial neural networks to predict application performance in virtualized environments. Further, Gulati et al. [3] present a study on storage workload characterization in virtualized environments, but perform no performance analysis.

The second group of related work deals with benchmarking and performance analysis of virtualized environments not specifically targeted at storage systems. Hauck et al. [12] propose a goal-oriented measurement approach to determine performance-relevant infrastructure properties. They examine OS scheduler properties and CPU virtualization overhead. Huber et al. [13] examine performance overhead in VMware ESX and Citrix XenServer virtualized environments. They create regression-based models for virtualized CPU and memory performance. In [14], Barham et al. introduce the Xen hypervisor comparing it to a native system as well as other virtualization plattforms. They use a variety of benchmarks for their analysis to quantify the overall Xen hypervisor overhead. Iyer et al. [15] analyze resource contention when sharing resources in virtualized environments. They focus on measuring and modeling cache and core effects.

7 Conclusions

We presented a detailed analysis of the performance-influencing factors of I/O-intensive applications in virtualized environments. Our analysis and evaluation is based on a real world deployment of the state-of-the-art virtualization technology of the IBM System z and the storage system IBM DS8700 used in a controlled testing environment. Our multi-tiered storage environment consists of storage caches and RAID arrays and is representative for any virtualized storage environment.

By analyzing our environment, we first systematically identified the relevant storage-performance-influencing factors of I/O-intensive applications in virtualized environments. We modeled the factors using hierarchical feature trees and organized them by workload, operating system, and hardware. The models were of general nature and not specific to the considered environment. Second, we proposed a generic workload and benchmarking methodology and applied it to our environment in order to quantify the impact of the relevant storage-performance-influencing factors. We showed especially high performance influence of the read/write ratio in the workload and the CFQ I/O scheduler in the system configuration. Finally, we effectively extracted performance models based on linear regression for predicting the performance of varying read/write requests. Throughput of requests was approximated very well. Regarding response time, sequential reads and big writes were approximated well for a cached file set and almost all requests were approximated well when the file set size was larger than the cache size.

Acknowledgments. This work was funded by the German Research Foundation (DFG) under grant No. RE 1674/5-1 and grant No. KO 3445/6-1. We especially thank the Informatics Innovation Center (IIC)[5] for providing the system environment of the IBM System z and the IBM DS8700.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM 53(4), 50–58 (2010)
2. Kraft, S., Casale, G., Krishnamurthy, D., Greer, D., Kilpatrick, P.: Performance Models of Storage Contention in Cloud Environments. Springer Journal of Software and Systems Modeling (2012)
3. Gulati, A., Kumar, C., Ahmad, I.: Storage workload characterization and consolidation in virtualized environments. In: VPACT 2009 (2009)
4. Mell, P., Grance, T.: The nist definition of cloud computing. National Institute of Standards and Technology 53(6), 50 (2009)
5. Dufrasne, B., Bauer, W., Careaga, B., Myyrrylainen, J., Rainero, A., Usong, P.: Ibm system storage ds8700 architecture and implementation (2010), `http://www.redbooks.ibm.com/abstracts/sg248786.html`
6. Wang, M., Au, K., Ailamaki, A., Brockwell, A., Faloutsos, C., Ganger, G.R.: Storage device performance prediction with CART models. In: MASCOTS 2004, pp. 588–595 (2004)
7. Bucy, J.S., Schindler, J., Schlosser, S.W., Ganger, G.R., Contributors: The DiskSim Simulation Environment - Version 4.0 Reference Manual. Carnegie Mellon University, Pittsburgh (2008)
8. Lebrecht, A.S., Dingle, N.J., Knottenbelt, W.J.: Analytical and simulation modelling of zoned raid systems. The Computer Journal 54, 691–707 (2011)
9. Ahmad, I., Anderson, J., Holler, A., Kambo, R., Makhija, V.: An analysis of disk performance in vmware esx server virtual machines. In: WWC-6, pp. 65–76 (2003)
10. Koh, Y., Knauerhase, R., Brett, P., Bowman, M., Wen, Z., Pu, C.: An analysis of performance interference effects in virtual environments. In: ISPASS 2007, pp. 200–209 (2007)

[5] `http://www.iic.kit.edu/`

11. Kundu, S., Rangaswami, R., Dutta, K., Zhao, M.: Application performance modeling in a virtualized environment. In: HPCA 2010, pp. 1–10 (2010)
12. Hauck, M., Kuperberg, M., Huber, N., Reussner, R.: Ginpex: deriving performance-relevant infrastructure properties through goal-oriented experiments. In: QoSA-ISARCS 2011, pp. 53–62. ACM, New York (2011)
13. Huber, N., von Quast, M., Hauck, M., Kounev, S.: Evaluating and modeling virtualization performance overhead for cloud environments. In: CLOSER 2011 (2011)
14. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. SIGOPS Oper. Syst. Rev. 37, 164–177 (2003)
15. Iyer, R., Illikkal, R., Tickoo, O., Zhao, L., Apparao, P., Newell, D.: Vm3: Measuring, modeling and managing vm shared resources. Computer Networks 53, 2873–2887 (2009)