IO Performance Prediction in Consolidated Virtualized Environments

Stephan Kraft*
SAP Research
The Concourse, Queens Road
Belfast, UK
stephan.kraft@sap.com

Giuliano Casale
Imperial College London
Dept. of Computing
London, SW7 2AZ, UK
g.casale@imperial.ac.uk

Diwakar Krishnamurthy
University of Calgary
Dept. of ECE
Calgary, AB, Canada
dkrishna@ucalgary.ca

Des Greer
Queen's University Belfast
School of EEECS
Belfast, UK
des.greer@qub.ac.uk

Peter Kilpatrick
Queen's University Belfast
School of EEECS
Belfast, UK
p.kilpatrick@qub.ac.uk

ABSTRACT

We propose a trace-driven approach to predict the performance degradation of disk request response times due to storage device contention in consolidated virtualized environments. Our performance model evaluates a queueing network with fair share scheduling using trace-driven simulation. The model parameters can be deduced from measurements obtained inside Virtual Machines (VMs) from a system where a single VM accesses a remote storage server. The parameterized model can then be used to predict the effect of storage contention when multiple VMs are consolidated on the same virtualized server. The model parameter estimation relies on a search technique that tries to estimate the splitting and merging of blocks at the the Virtual Machine Monitor (VMM) level in the case of multiple competing VMs. Simulation experiments based on traces of the Postmark and FFSB disk benchmarks show that our model is able to accurately predict the impact of workload consolidation on VM disk IO response times.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques

General Terms

Performance, Measurement, Management

Keywords

Queueing Networks, Simulation, Virtualization, Storage Contention

*Stephan Kraft is also affiliated with Queen's University, Belfast, UK.

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.

1. INTRODUCTION

The performance of IO-bound applications is dominated by the time required by the operating system to schedule read and write operations and by the response times of the storage devices in completing such requests. Since changes in the workload, as well as in the software and hardware environments, can affect the latency of disk IO requests, it is often useful to define performance models to anticipate the effects of a change. This is especially important in virtualized data centers, where the concurrent shared use of a storage device by several Virtual Machines (VMs) managed by a Virtual Machine Monitor (VMM) can lead to significant performance degradation [25]. In such systems estimates of IO contention for a given VM placement configuration can support management and consolidation decisions. However, modeling the performance of disk requests is very challenging due to the joint interaction of the IO flows issued by several VMs and because of the complexity of caching mechanisms, scheduling algorithms, device drivers, and communication protocols employed by both the VMs and the VMM.

In this paper, we tackle this complexity by introducing a trace-driven simulation approach for IO performance prediction in consolidated environments where multiple VMs can share access to a remote storage server. Our methodology, summarized in Figure 1, requires first to study VMs when they run in isolation on a virtualized server. After collecting measurements, we use simulation and specialized parameterization techniques to forecast the impact of consolidation on IO performance. Specifically, for each VM of interest we collect traces of arrival times and estimated service times for IO requests. We then use these isolation scenario traces to parameterize a queueing model for a specified consolidation scenario. A feature of our queueing model is also to use start-time fair queueing (SFQ), a popular implementation of fair share scheduling which is adopted in VMMs.

Motivated by the fact that a trace-driven simulation that only uses such arrival and service time traces typically *fails* in predicting accurately IO request response times under consolidation, we define an iterative algorithm for optimal parameterization of the simulation model. Specifically, the algorithm employs a search technique to estimate the performance impact of VMM level IO optimizations such as the

splitting and merging of IO requests. Extensive experimentation on test workloads generated with the Postmark (PM) and FFSB disk benchmarks reveals that our methodology can forecast successfully consolidation effects on IO performance.

Summarizing, the main contributions of this paper are twofold.

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

2. The model is enhanced with an iterative technique that estimates the impact of optimization operations performed by the VMM kernel disk scheduler, such as merging of requests.

The remainder of the paper is organized as follows. Section 2 motivates the use of prediction tools to quantify performance degradation of disk requests in shared environments and Section 3 introduces the reference system for our study. The proposed modeling methodology is presented in Section 4 and model validation results are shown in Section 5. Section 6 gives an overview of related work. Section 7 offers summary and conclusions.

2. MOTIVATIONAL EXAMPLE

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

In consolidation scenarios where more than a single VM submits large numbers of IO requests, competition for the disk drive can lead to significant increases in latencies, i.e. response times. To illustrate the problem Figure 2 compares the distribution of response times in two experiments conducted on our reference system, which is introduced in Section 3.1. In the first experiment, denoted isolation experiment, a single VM runs on the system thus avoiding contention from other VMs. In the second set of experiments,

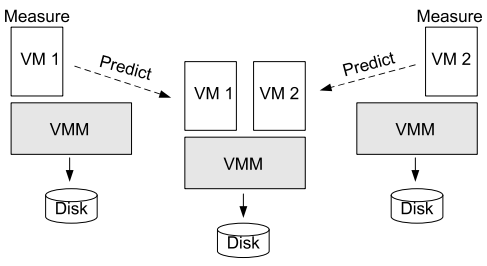


Figure 1: Problem Approach.

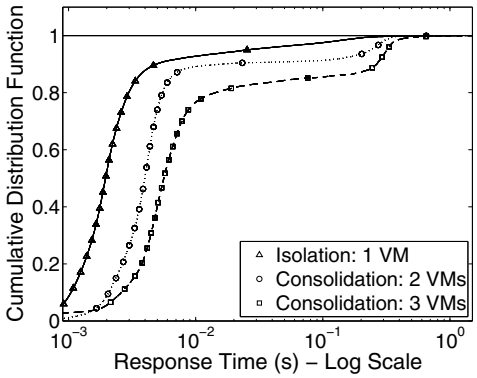


Figure 2: Effect of VM Consolidation on Disk Request Latencies.

denoted consolidation experiment, we run two or three identical VMs on the same virtualized server and find that their disk requests experience storage contention. Response times are computed as the time between request issue and request completion as recorded by the VM operating system. In isolation the CDF in Figure 2 shows that the 95th percentile of response times is 0.0255s. Moving from the isolation case to a consolidation scenario the probabilities of observing small response times remarkably decrease. For the experiments where two and three VMs submit an identical workload to the same storage device, we recorded 95th percentiles for the storage response times of 0.238s and 0.3198s, respectively. This makes a strong case for the significant performance impact that consolidation can have on end-to-end response time of IO requests and motivates the investigation in this paper. Specifically, the example motivates the need for accurate models that can capture and quantify the possible IO performance degradation related to consolidation effects.

3. SYSTEM CHARACTERISTICS

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

3.1 Reference System

We conduct our study on an AMD-based enterprise server with 4 quad-core processors containing a total of 16 CPU cores each clocked at 2.21GHz. The system is equipped with 68GB of RAM and is connected to an OpenFiler [3] storage server via the iSCSI protocol and 1 GBit Ethernet. The storage server manages a SATA-II hardware RAID controller which in turn manages a 15 disc RAID 5 array.

On the server we host the virtualization platform VMware ESX Server 3i - 3.5.0 [7], which accesses the storage device through a software iSCSI host bus adapter (HBA). The virtualized environment consists of multiple Debian 5.0.1, kernel 2.6.26-2, guest VM systems, each with identical configurations of 1 virtual CPU, 500MB RAM, and 50GB of "virtual" hard disk formatted as *ext3* file system.

The virtual disks are represented by a large file and can be thought of as a linear array made up of units of space, i.e. logical blocks. In the remainder of this paper the term

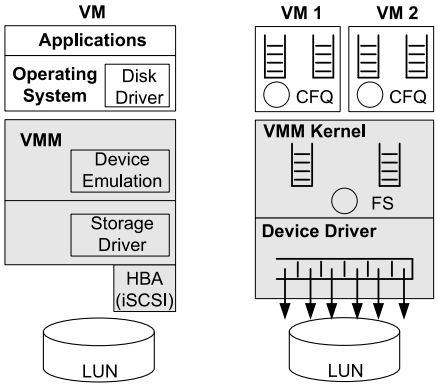


Figure 3: IO Architecture and Storage Queues of the Reference System.

"block" always refers to logical block units, rather than the storage device's physical block units. In our system virtual disk files of all VMs are stored on a single physical storage device (LUN) and access the LUN through the same HBA.

Disk IO requests issued from a VM consist of one or multiple contiguous blocks for either reads or writes. Once an application running inside the VM submits a request, the request first goes to the disk driver of the VM operating system as shown in Figure 3 (a). The driver processes the request and forwards it to the VMM, where it is trapped and may undergo further optimization operations before being issued to the LUN via the storage driver and the iSCSI HBA [9].

On their way through the previously described layers of the storage subsystem, requests can be queued multiple times as illustrated in Figure 3 (b). Hence latencies of requests may be affected by multiple queueing delays. Furthermore, requests may undergo optimizations such as splitting, merging, and reordering by schedulers managing the various queues shown in Figure 3 (b). Such operations are used to optimize disk access patterns, e.g., by merging multiple requests for small amounts of contiguous blocks to fewer requests for large amounts of contiguous blocks. In virtualized environments these operations can have a significant impact on disk request performance, as scheduling policies at VM and VMM level may impair each other [14].

In our environment the Debian guest VMs are configured to use the completely fair queueing (CFQ) [10] scheduler, which has per default 64 internal queues to maintain and keep disk IO requests [5]. The in-VM scheduler optimizes disk access for the ext3 file system, which is configured at the default block size of 4kB. Hidden from the operating system of the VM, the VMM conceptually comprises two sets of queues, namely the VMM kernel queues and the device driver queue. The VMM kernel maintains a queue of pending requests per VM for each target SCSI device [21], controlled with a fair-share (FS) [19] scheduler. See section 4.1 for more detailed information on the scheduling. The VMM formats virtual disk files in the *virtual machine file system* [7] and performs its own splitting, merging and reordering processing on disk requests that are queued at the VMM kernel.

Furthermore, the server maintains a device driver queue for each LUN, which controls the *issue queue length* defined as the number of pending requests the server can have at the storage device at a time [6]. Virtualized servers typically allow to configure the issue queue length for each LUN. When multiple host servers issue requests to the same LUN, this parameter can be used to control resource utilization and fairness across hosts.

Summarizing, our reference system comprises a network of interconnected queues with multiple classes of customers corresponding to the multiple VMs and various scheduling disciplines, which need to be represented in the performance model. In this case the application of classic solutions for product form queueing networks [11] is complicated due to complex splitting, merging, and reordering operations on arriving requests that depend on spatial locality on the storage device. Although forking and joining operations may alleviate such difficulties, they are hard to parametrize in absence of direct measurement of the VMM internal optimization, which is the typical situation in practice. Furthermore, the merged submission of requests can result in an extreme behaviour of the arrival patterns where large amounts of requests are condensed into large bursts. Bursts have been identified as important sources of performance degradation [28, 16] and we handle them in this work by resorting to trace-driven simulation.

3.2 Measurement Tool

This section describes the monitoring tool we have used to quantify the performance of disk requests, as well as to collect necessary data for parameterization of our model. All measurements are obtained inside VMs with the block layer IO tracing mechanism *blktrace* [1]. The tool allows us to record traces of disk request *issue* and *completion* events, as they are recorded from the VM operating system kernel. *Issue* events specify that a request that has previously resided in the disk IO scheduler queue of the VM operating system has been sent to the driver, while *completion* events mark that a previously issued request has been completed [15]. Collecting this data at the operating system level, rather than at the level of the in-VM application, has the advantage that sorting and merging operations of the VM CFQ scheduler are reflected in the monitoring trace.

Traces comprise a relative time stamp for each request issue and completion, a flag that indicates whether a request was a read or write, the logical block address (LBA) pertaining to the request, and the number of blocks accessed. This detailed information is helpful for building the system model, since it enables us to compute per request arrival times at the VMM. In order to capture request splitting and merging operations at the VMM kernel level we also record the block size of each request arrival. Specifically, the VMM may split requests above a certain size threshold. It may also merge several requests together to leverage spatial locality. We note that we calculate per request response times as the time difference between completion and issue events in the trace collected using *blktrace*.

4. MODELING METHODOLOGY

Our model comprises trace-driven simulations to predict the performance degradation of VM disk request response times due to storage device contention. The fundamental idea of our approach is to first record application traces in isolation benchmark experiments and then utilize these traces to parametrize simulation models of consolidation sce-

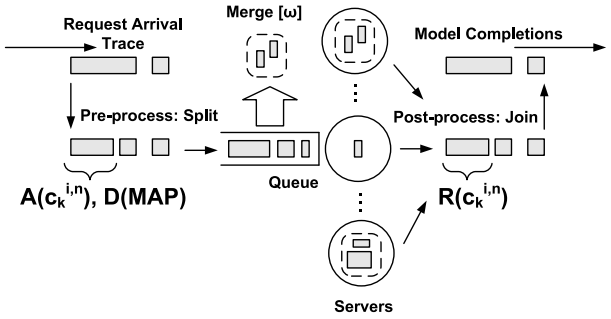


Figure 4: Methodology Overview.

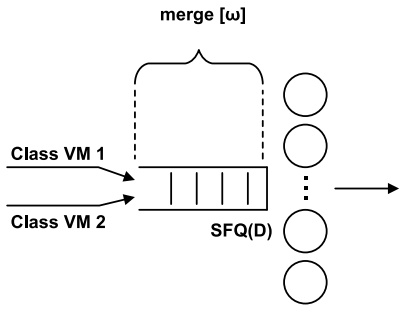


Figure 5: Queueing Model.

narios. The challenge is to define realistic models of contention and of the internal VMM IO optimizations which affect the end-to-end delays of read and write operations. Model parameterization is solely based on measurements obtained within the VMs and information that we gathered from available documentation on VMM operations such as fairshare scheduling, splitting and merging. As a result, we essentially treat the VMM as a blackbox in this work.

Figure 4 shows an overview of our methodology and introduces some of the terminology used in the following sections. The model allows us to study a specified consolidation scenario consisting of a set of VMs concurrently sharing a storage device. Input parameters for the queueing model are request arrival traces obtained from in-VM measurements in isolation benchmark experiments for each of the VMs considered in the consolidation scenario. Our methodology can account for environment specific, VMM level request splitting behavior by means of a pre-processing step on arrival traces, as described in Section 4.2.1. Besides having an arrival time $A(c_k^i)$ each request c_k^i is also provided with a service time $D(c_k^i)$ sampled from a Markovian Arrival Process (MAP) as described in Section 4.2.2. As shown in Figure 4, we use a simulation model that schedules requests using the SFQ(D) [23] scheduling policy across a pool of servers, each representing a parallel connection to the LUN. This model additionally approximates the request merging behavior of the VMM disk scheduler by bundling a configurable number of ω requests and enabling them to share a server. The queueing network and the scheduling algorithm are presented in 4.1. The search algorithm we developed to iteratively estimate the parameter ω is introduced in Section 4.3.2. Finally, the model outputs requests with a response time estimate $R(c_k^i)$. This involves a post-processing task wherein, as shown in Figure 4, the requests that have been split in the pre-processing step are rejoined.

4.1 Simulation Model

We represent the system under study as a multiclass open queueing model. Requests submitted from individual VMs are distinguished in separate classes as shown in Figure 5. As described in Section 3 virtualization environments typically provide a configurable parameter which controls the maximum VMM issue queue length to the LUN. We capture this aspect by modeling the storage device as a pool of parallel servers. In our reference system this parameter is maintained at its default value of 32 and consequently our model comprises of 32 servers.

Based on available documentation on the VMM, we implemented a practical SFQ disk scheduling discipline to schedule requests on to the servers. Fair queueing [19] algorithms are work-conserving and schedule shared resources between competing requests by allocating resource capacity based on proportional weights. Practical fair queueing algorithms constitute approximations to generalized processor sharing (GPS) scheduling by considering requests to be indivisible, rather than fluid flows.

Conventional SFQ schedulers do not consider concurrent service of requests at a resource. Our simulation model implements SFQ(D), a special variation of SFQ [20], which has previously been used in related work [21] to model our reference system. The depth parameter D controls the number of concurrent requests in service and consequently corresponds to the number of servers in our model. Upon arrival of the i^{th} request c_k^i of class k, it is assigned a *start tag* $S(c_k^i)$ and a *finish tag* $F(c_k^i)$ by the scheduler. The tag values represent the times at which each request should start and complete service according to a system notion of *virtual time* $v(t)$. Tags are computed as:

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

$$F(c_k^i) = S(c_k^i) + \frac{d_k^i}{\phi_k}, j \geq 1 \quad (2)$$

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

The scheduler issues a maximum of D requests to idle servers in increasing order of *start tags*. When a request completes the queued request with *min(start tag)* is selected and issued to an available server to maintain a concurrency level of D. *Virtual time* advances by assigning it the start tag of the last request issued on or before time t, i.e., the queued request with the lowest start tag at the time of the last issue. As mentioned previously, in addition to SFQ(D) scheduling we also select ω requests, merge them together, and issue the merged request to the servers. The superposition of this behaviour with SFQ(D) is described in Section 4.3.2.

4.2 Model Parameterization

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

4.2.1 Interarrival Times

The simulation model is parameterized with measured arrival traces, which are recorded in a series of benchmark experiments. Benchmark workloads are submitted from within

VMs running in isolation, where only a single VM is running on the server. For each VM we build a trace repository comprising multiple benchmark runs. Traces are recorded with the *blktrace* tool, where we include every in-VM request issue as an arrival in the model.

When predicting request response times for consolidation scenarios, we randomly choose an arrival trace for each considered VM from the repository and run a consolidation simulation. Parameterizing the simulation model with arrival traces measured in isolation experiments is valid, since our measurements in Section 5.2 show that the interarrival time distribution of disk requests to the VMM is not significantly impacted by workload consolidation. This indicates that in the presence of contention delays disk requests are queued at the VMM, rather than at the VMs. Queueing requests at the VMM is preferable, since queue depths should be larger than at the VMs and the VMM can use system specific information to optimize disk access of queued requests. We expect this observation to hold unless the VMM queues are saturated.

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

$$N_k^i = \left\lceil \frac{B(c_k^i)}{l_{max}} \right\rceil \tag{3}$$

$$A(c_k^{i,n}) = A(c_k^i) \tag{4}$$

$$B(c_k^{i,n}) = \begin{cases} l_{max} & \sum_1^n B(c_k^{i,n}) \leq l_{max} \\ B(c_k^i) \bmod l_{max} & \sum_1^n B(c_k^{i,n}) > l_{max}, \end{cases} \tag{5}$$

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

$$R(c_k^i) = \frac{1}{N_k^i} \sum_{n=1}^{N_k^i} R(c_k^{i,n}). \tag{6}$$

As requests are likely to be served in parallel by the storage array, computing the mean response time of split requests can only be an approximation of the real behaviour at the device. We investigate the effectiveness of this approximation in Section 5. In our reference system the VMM splits arriving requests exceeding a size of 512kB, corresponding to 128 blocks in an ext3 file system with 4kB block size. We consider this behavior in our model and set $l_{max} = 128$.

4.2.2 Service Times

Service times are key parameters to specify queueing models and are typically estimated based on direct data measurement and statistical inference. A common approach to characterize the resource consumption of requests is to monitor system utilization and use regression techniques based on operational laws [27]. As our blackbox approach does not involve instrumentation of the VMM or of the storage server in order to collect utilization samples, a practice which is anyway difficult or impossible in many real-world systems, we approximate the service times of disk requests from response time measurements in isolation benchmark experiments. When utilization levels at the VMM are low, disk requests do not face queueing delays as they get instantly issued to the storage device. In fact, our measurements show that the mean number of requests in the system during an isolation run does not exceed the number of available connections from the VMM to the LUN. Thus measured request response times in isolation should be a reasonable approximation to the actual service requirement at the disk. Cases where this approach is problematic involve load-dependent service times, however the experiments reported in this paper indicate that such effects may not be prevalent in the consolidated environments we consider.

Request service times belonging to a specific VM collected during isolation experiments are fitted to a MAP [17], which is then used to randomly sample a service time for each arrival. The role of MAPs in our methodology is to generate random traces, which follow the same statistical properties (distribution, autocorrelations) observed in the isolation experiments.

4.3 Request Merging Methodology

Our simulation experiments indicate that prediction results can be significantly increased if the model is enhanced with heuristics to account for the merging operations done by the VMM. As we mentioned previously, we allow the scheduler to merge queued requests. The merging algorithm is described in Section 4.3.1. The iterative technique to quantify the amount of performed scheduler merging operations for a given workload configuration is described in Section 4.3.2.

4.3.1 Request Merging Algorithm

The model merges a configurable amount of queued requests in such a way that they still remain separate entities, but share a server, i.e, connection to the LUN. As shown in Algorithm 1, we pass a merge value parameter, denoted ω, to the simulator, which approximates the amount of merging operations observed in the real system. Merging values can range in $[1, \infty]$, where we consider a merge value of 1 as a single job being issued per service station, i.e. no merging, whereas a large ω indicates that several requests are merged together before issue to the storage server. Since requests are indivisible, we implement a function *get_int_value* to obtain from ω the number of merged requests in each round of VMM IO optimizations. For example, if ω is configured as 2.5 there is an equal probability that the maximum amount of merging operations performed next by the simulator will be either two or three.

The technique maintains the properties of the SFQ scheduler by merging requests in increasing number of start tags. Furthermore, only requests of the same class are merged. As a result, the algorithm aborts in cases where the queued request with the minimum start tag is of a different class as the already merged requests. Once a merged job has received service and exits the model, each of the merged re-

Algorithm 1 Implementation of Merging

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

quests counts as a separate completion. Since each of the requests sharing a service station has an individual service time, we approximate the aggregate service requirement of merged request with the mean of the individual request service times.

4.3.2 Merge Value Estimation

The challenge is to approximate the merging operations inherent in the real system without detailed knowledge of the VMM internals and further complication of the simulation model, e.g. by explicitly considering spatial locality of requests in the simulator. To estimate the merging value in this blackbox view of the VMM we have developed an iterative search technique. Our technique controls the mean number of requests in simulation experiments, i.e. the mean queue length, through the ω parameter and terminates once the mean queue length seen in simulation closely matches an inferred *expected* queue length.

Inference of Mean Expected Queue Length. The first step of merge value estimation is to infer the expected mean queue length in the system for the consolidation scenario under study. We infer the expected mean queue length for a consolidation scenario with K classes based on the assumption that the mean number of requests in the system grows linearly when moving from isolation to consolidation:

$$N_{exp}^{K} = \sum_{i=1}^{K} N_{meas}^{iso}, \tag{7}$$

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

To validate this linear assumption Table 1 shows measurements of the mean number of requests from benchmark experiments in our reference system. We present measurements for a number of workload configurations averaged over multiple benchmark runs. For detailed information on the considered workloads see Section 5.1. Results indicate that the linear assumption is a good approximation to system behavior. We expect our assumption to hold as long as the aggregate mean number of requests outstanding from VMs, i.e. requests issued and not yet completed, does not exceed the number of available connections from the VMM to the LUN.

Table 1: Measurements and Expected Mean Number of Requests N in the System in Isolation (Iso) and Consolidation Scenarios with Two VMs (Con 2) and Three VMs (Con 3)

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

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

$$N'_{sim} = \frac{N_{sim}}{\omega},\ \omega \geq 1 \tag{8}$$

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

$$\Delta_{\omega} = \left| \frac{N'_{sim} - N_{exp}}{N_{exp}} \right|, \tag{9}$$

where N_{exp} is the inferred expected queue length computed according to (7) from isolation experiments. The search terminates if the corrected queue length is accurate within 5% of N_{exp}. In cases where the estimation error is outside of this range, we control search direction and ω values on basis of a binary search. Let $\omega = g(N'_{sim})$ be the merge value used in simulation to obtain N'_{sim}. If N'_{sim} is smaller than N_{exp}, we decrease ω in order to increase the mean number of request in simulation. In cases where previous iterations have produced a $N'_{sim,old}$, such that $\{N'_{sim,old} > N_{exp} > N'_{sim}\}$ the merge value for the next iteration is determined by

$$\omega = g(N'_{sim}) - \frac{g(N_{sim})' - g(N'_{sim,old})}{2}, \tag{10}$$

which is half the distance between ω values used to obtain

Algorithm 2 Iterative Estimation of Merge Value

```
ω ← merge value initialisation point
N_exp ← infered expected queue length
Δ_max ← 0.05
flag ← 0
while flag < 1 do
  #run configurable amount of simulator iterations
  for i = 1 to max_simulator_iterations do
    for k = 1 to K do
      draw random arrival trace from repository
    end for
    simulate(ω)
  end for
  #search merge value ω
  N_sim ← mean queue length over simulator iterations

  N'_sim ← (N_sim/ω)
  Δ_ω ← get_merge_error(N'_sim)
  if Δ_ω ≤ Δ_max then
    flag ← 1
  else if N'_sim < N_exp then
    ω ← decrease
  else if N'_sim > N_exp then
    ω ← increase
  end if
end while
```

N'_{sim} and $N'_{sim,old}$. In cases where no such $N'_{sim,old}$ exists, we decrease ω by a configurable step size parameter. The inverse of the above applies for the opposite search direction.

5. VALIDATION EXPERIMENTS

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

5.1 Workload Generation

We consider a number of different workload types, which are submitted from within VMs running in isolation, as well as in consolidated scenarios. The workloads consist of varying configurations of two benchmarks with quite distinct characteristics. The Flexible File System Benchmark (FFSB) [2] is a multi-threaded benchmark comprising large file operations. Conversely, Postmark (PM) [4] is a synthetic single-threaded workload comprising small file and metadata-intensive operations designed to emulate IO patterns of Internet applications such as e-mail and e-commerce.

We specifically choose these two workload types to validate our model, since the extent of optimization operations of the VMM should significantly differ in both cases. The file operations of FFSB include disk requests which consist of large numbers of contiguous blocks. Here, the VMM scheduler might not have much opportunity to perform further merging operations and may even have to split requests that are too large. PM requests, on the other hand, consist of small amounts of blocks and may allow the disk scheduler to merge to a larger extent.

In order to obtain a system steady state, benchmarks for each configuration are submitted over a period of 75 minutes in five minute intervals. While the FFSB application can be parameterized to run for a specified time frame, PM does not provide such an option. In cases where a PM run needs less than five minutes to complete we restart it, but only collect monitoring data within the five minute time window that captures a run.

Table 2: Workload Configurations.

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

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

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

FFSB. Similar to PM we have defined two FFSB configurations, but this benchmark additionally supports the execution of *sequential*, as well as *randomized* reads/writes. The response times of sequential and random requests can significantly differ, since sequential requests are most directly affected by the transfer speed of the disk drive, while random requests are most directly affected by disk seek time [5]. Sequential read requests can also take advantage of *read-aheads*, which assume that an application reading from disk block n will next ask to read from the subsequent disk blocks. The read-ahead mechanism decreases seek times by caching the following disk blocks $n+1, n+2, etc.$, into memory, but is turned off once the system detects a non-sequential file access. Considering the sequential and randomized options essentially leaves us with four distinct FFSB workloads for our study, denoted as FFSB-1_S, FFSB-1_R, FFSB-2_S, and FFSB-2_R.

At the beginning of each benchmark run FFSB dedicates a single thread to create an initial file set for the subsequent operations. In our specific configuration the application creates a structure of 400 files and 10 directories. Even though both configurations are initialized with the same total number of files, the distribution of file sizes is different. FFSB allows us to specify the quantity of certain file sizes, as well as the probability of IO operations via relative weights. The file sizes in FFSB-1_S/R and FFSB-2_S/R range in [4kB;32MB] and [32kB;32MB], respectively, with the highest request frequencies for the smaller sizes. We de-

fine a single threadgroup consisting of 100 threads and set the think time parameter to zero. Read and write IO operations are configured with the same weight, but different sizes, as shown in Table 2.

5.2 Workload Characterization

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

5.2.1 Request Size

To illustrate the different size characteristics of the workload configurations considered, Figure 6 shows measurements of VM disk request size distributions of sample benchmark experiments. For ease of presentation we have grouped the data into bins. The sizes of the bins are chosen on the basis of a related workload characterization study [8]. As shown in Figure 6 (a) the VM kernel merges the majority of requests submitted by PM-1 into sizes larger than four and less or equal to eight blocks. Since the standard file system block size on our reference system is 4kB, a request for eight blocks has a size of 32kB. Figure 6 (b) reflects the main difference between the two PM workload configurations. PM-2 submits requests of larger sizes. This results in a lower frequency of 32kB (eight blocks) request sizes and additional requests for block sizes greater than eight. A common attribute of the two PM workloads is that neither workload causes the VM scheduler to issue a significant number of requests with blocks sizes greater than 128.

Figures 6 (c) and (d) show request size distributions of FFSB workloads and reveal that workload characteristics are quite different compared to PM workloads. Similar to PM the VM scheduler merges a large number of the FFSB workload to requests of size 32kB (8 blocks). However, the total number of requests is significantly lower and the proportion of requests with large sizes is significantly higher than in the case of PM. Evaluating the main differences between FFSB-1_S and FFSB-2_S, the increased file sizes and frequency of large file operations of FFSB-2_S allow the scheduler to translate FFSB-2_S workloads into fewer requests of larger sizes as seen in Figure 6 (d). The total number of requests compared to FFSB-1_S decreases, while the frequency of requests of block size 64 and above increases. For both FFSB workloads large proportions of requests are merged to block sizes > 128, which corresponds to request sizes $> 512kB$.

Summarizing, we have shown that sizes of requests submitted from VMs to the VMM significantly differ from the configured request sizes of the benchmark applications. Furthermore, PM workloads comprise large amounts of small requests that may be further merged when queued in the VMM kernel. In contrast, requests of FFSB workloads are already merged to large sizes by the VM disk scheduler and this might not allow the VMM kernel the same leverage with respect to the extent of merging operations as in the case of PM.

Table 3: Mean in ms, Standard Deviation (std), and Coefficient of Variation (cv) Statistics for Request Interarrival and Service Times.

Workload	*Interarrival Times*			*Service Times*		
	mean	std	cv	mean	std	cv
PM-1	0.72	0.02	21.0	8.61	0.04	5.0
PM-2	0.81	0.01	16.4	11.5	0.09	5.1
FFSB-1_S	3.07	0.03	9.2	13.9	0.05	3.4
FFSB-1_R	3.24	0.03	9.1	15.6	0.05	3.1
FFSB-2_S	4.36	0.04	9.9	15.2	0.05	3.4
FFSB-2_R	3.54	0.04	10.8	14.1	0.05	3.6

5.2.2 Interarrival Times

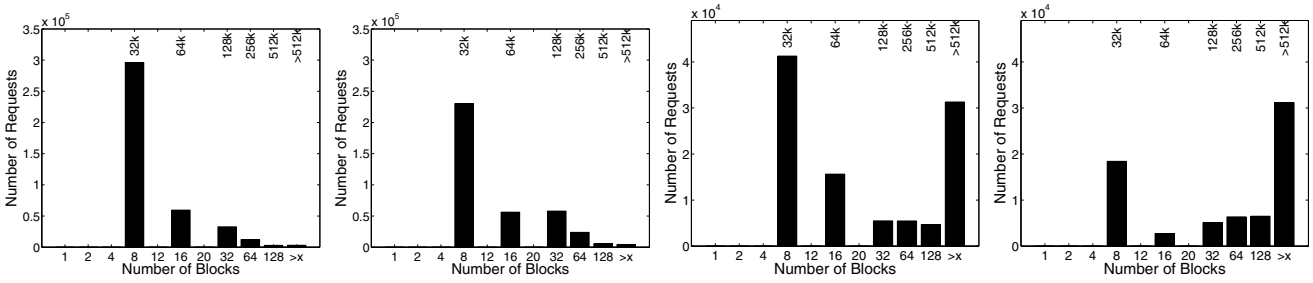
The significantly different characteristics of the considered workload configurations are also reflected in the distributions of request interarrival times. Figure 7 (a) shows the interarrival time distributions of requests to the VMM, as recorded when submitting the PM workloads. While both distributions have a similar shape, there is a higher probability for a longer interarrival times in PM-2. We computed the mean interarrival times for PM-1 and PM-2 as 0.72ms and 0.81ms, respectively, as shown in Table 3.

Figure 7 (b) illustrates the arrival distributions for both FFSB configurations when performing sequential read/write operations. Compared to the PM workloads, the interarrival times are considerably longer. We observe a significantly higher probability for longer interarrival times for FFSB-2_S with a 95th percentile of 13.6ms. FFSB-1_S has a shorter mean interarrival time across the whole range of the distribution with the 95th percentile of interarrival times being 8.5ms or less. The distribution of arrivals does not seem to be affected by randomizing disk access, as Figure 7 (c) indicates.

Since our model uses information recorded during isolation benchmarks experiments only, we are especially interested in quantifying the impact of workload consolidation on request interarrival times. Figure 8 shows the interarrival time distributions of disk requests submitted from VMs when running in isolation (Iso) compared to consolidation scenarios with additional VMs submitting an identical workload in the background. Interestingly none of the arrival distributions from the VM to the VMM displays a large deviation from their corresponding isolation distributions when the workload on the server is doubled (Con 2) or tripled (Con 3). We take this as a clear indication that queueing of disk IO requests due to resource contention takes place at the VMM layer, rather than at the VMs themselves.

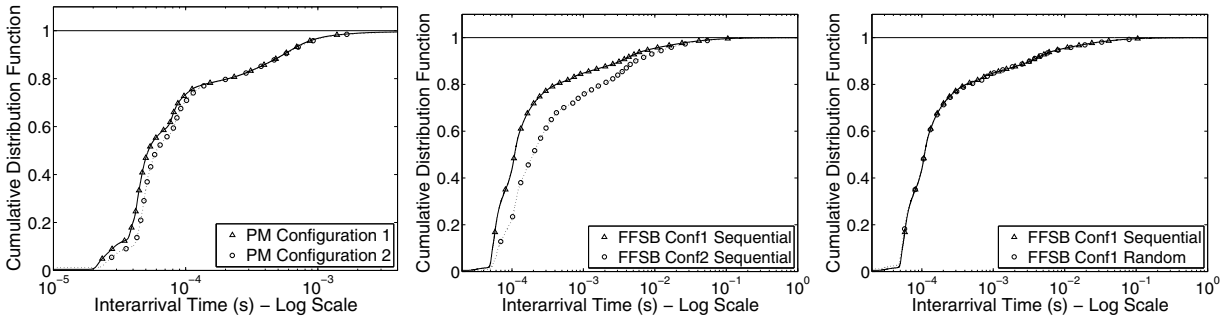
5.2.3 Service Times

Our methodology approximates the service requirement of disk requests with measured response times in isolation scenarios as described in Section 4.2.2. In isolation the utilization levels at the VMM are likely to be low and thus requests may not incur queuing. Table 3 shows mean service time statistics for all workload configurations averaged over all VMs and benchmark runs. The service requirements for FFSB workloads are higher than for PM. This is probably due to the larger request sizes of FFSB which entail



(a) PM-1 (b) PM-2 (c) FFSB-1_S (d) FFSB-2_S

Figure 6: Measured Distribution of Request Sizes for PM and FFSB Workload Configurations.



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

Figure 7: Distribution of Arrival Processes of PM and FFSB Workloads in Isolation.

increased lengths of read/write operations at the disk drive. Interestingly, randomizing workload patterns does not automatically lead to higher service times. FFSB-1_R has a larger service time than FFSB-1_S, while it is the opposite for FFSB-2_R and FFSB-2_S.

5.3 Model Validation

In this section we study the accuracy of our proposed methodology to predict the mean response time of disk requests when consolidating storage workloads. For validation we conduct a series of benchmark experiments on our reference system using the previously introduced workload configurations. Workloads are submitted from within VMs. We consider scenarios with up to three VMs, denoted VM 1, VM 2, and VM 3, where we consolidate homogeneous workloads, i.e., workloads of the same type. Each benchmark experiment consists of 15 individual 5min runs and results are reported as means over all runs. We first show measurement results before comparing the prediction accuracy of our simulation model to a product form analytical solution.

5.3.1 Measurement Results

We present measurements that illustrate the impact of workload consolidation on disk request response times. Disk response times are an important metric since they may directly affect the performance of end users of applications, e.g. in cases where processes block until completion of a read request. All results are based on in-VM measurements obtained with the *blktrace* tool as described in Section 3.2.

Table 4 shows that workload consolidation leads to an increase of disk IO response times across all workload configurations. The PM workloads suffer a higher performance degradation than FFSB, with an increase ranging in approximately [258%; 299%] and [399%; 568%] in the two and three VM consolidation scenarios, respectively. In case of the FFSB workloads this increase is less severe, but still ranges approximately in [126%; 141%] and [147%; 181%] over all VMs and configurations. Furthermore, there is no clear trend showing that response times of random FFSB workloads increase to a larger degree than sequential FFSB workloads when consolidating. Interestingly, VMs seem to have slightly different IO characteristics even for identical workload configurations. For example, in a three VM consolidation scenario PM-2 requests submitted from VM 2 have a mean response time of 56ms, while the mean response time of PM-2 requests submitted from VM 3 is 43.9ms.

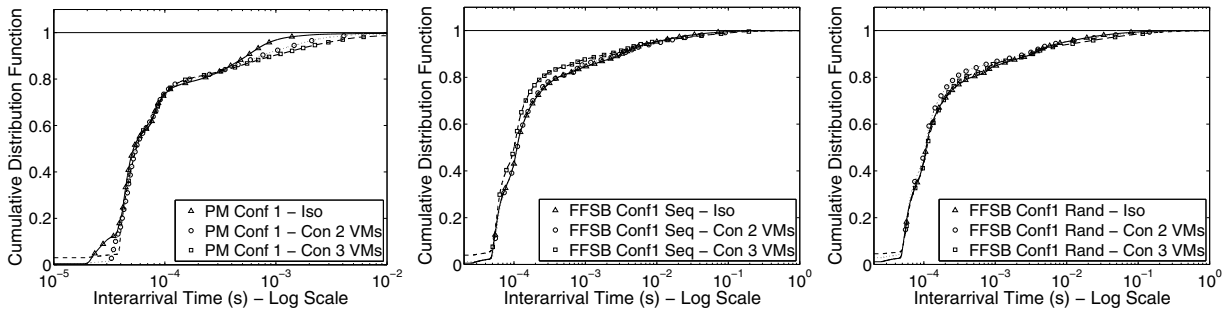
5.3.2 Prediction Accuracy

In this section we present predictions of mean disk request response times obtained using the proposed methodology. We validate model predictions against system measurements. We then compare the quality of our predictions to an open product form solution, which has previously been successfully applied in environments with shared resources [13]. Response time estimates for the product form model are determined analytically as

$$R_{est,k} = \frac{D_k}{1 - \sum_{t=1}^{K} \frac{\lambda_t}{n} \times D_t}, \tag{11}$$

where $R_{est,k}$ is a response time estimate for class k requests, D is the mean service time, λ the mean arrival rate, K the number of request classes, and $\{n = 32\}$ the number of servers in the model. In case the sum in the denominator equals a result ≥ 1, we set the value of the summation to 0.99. This term stands for the server utilization and may be affected by error due to our approximations on service demand estimates.

Predictions of our simulation model are averaged over multiple simulation runs, with number of runs ≥ 50 and ≥ 100 for PM and FFSB simulations, respectively. We indicate the reliability of the estimate using a 95% confidence



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

Figure 8: Impact of Workload Consolidation on Arrival Processes of PM and FFSB Workloads.

Table 4: Mean Response Time Measurements of Disk IO Requests in ms for VMs in Isolation (Iso) and Consolidation Scenarios with Two VMs (Con 2) and Three VMs (Con 3).

Workload	*VM 1*			*VM 2*			*VM 3*	
	Iso	Con 2	Con 3	Iso	Con 2	Con 3	Iso	Con 3
PM-1	9.45	28.3	47.7	8.49	25.4	48.2	7.9	37.8
PM-2	11.5	31.1	53.3	11.9	30.8	56.0	11.0	43.9
FFSB-1_S	14.8	18.4	24.9	13.5	16.8	24.5	13.3	25.2
FFSB-1_R	16.4	21.5	29.8	15.2	20.1	25.6	15.3	28.2
FFSB-2_S	15.4	19.9	24.9	15.2	20.4	22.3	15.1	23.6
FFSB-2_R	14.6	19.6	23.5	13.7	19.4	22.9	13.8	23.3

Table 5: Confidence Interval Width of Simulation Results (CI_β) and Mean Relative Errors of Response Time Predictions for Simulation (Δ_ω) and Product Form (Δ_p) Model.

Workload	*Con 2*			*Con 3*		
	CI_β	Δ_ω	Δ_p	CI_β	Δ_ω	Δ_p
PM-1	11.6	0.13	0.46	17.9	0.09	18.4
PM-2	8.9	0.08	2.26	10.3	0.06	64.9
FFSB-1_S	2.3	0.03	0.12	12.3	0.02	0.06
FFSB-1_R	4.2	0.03	0.09	27.7	0.30	0.03
FFSB-2_S	6.6	0.09	0.03	19.4	0.04	0.05
FFSB-2_R	1.8	0.16	0.03	5.05	0.15	0.04

Table 6: Comparison of Merging and Splitting Operations Performed by the Simulation Model.

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

interval and report the confidence interval width CI_β as the percentage of the mean, averaged over all classes. Furthermore, the model incorporates some specific characteristics of our reference system. The VMM splits incoming traffic above a size threshold of 512kB (128 blocks), which we consider in the parameterization of the model as described in Section 4.2.1. The quantity of splitting operations is reported as

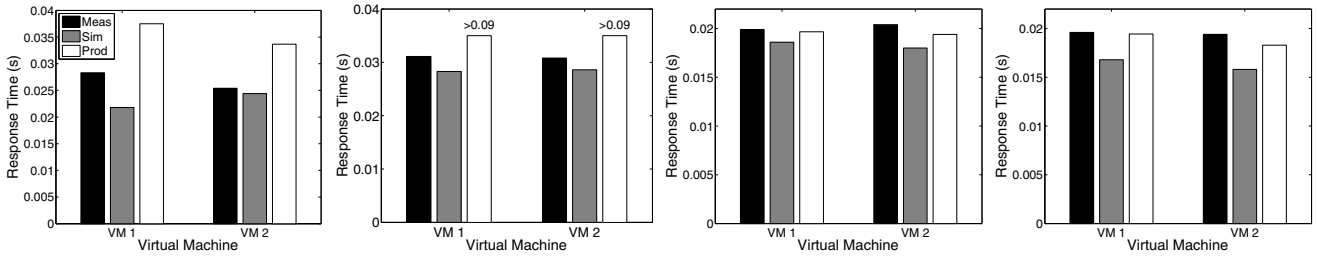
$$\Psi = \frac{C^I_{split}}{C^I}, \tag{12}$$

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

$$\Delta = \sum_{k=1}^{K} \frac{1}{K} \left| \frac{R_{est,k} - R_{meas,k}}{R_{est,k}} \right|, \tag{13}$$

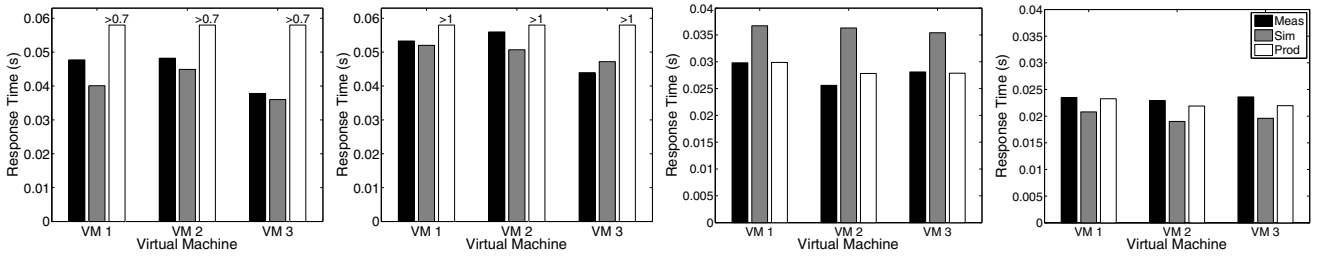
which is the mean relative error over all k classes of the estimated response time $R_{est,k}$ with respect to the measured value $R_{meas,k}$.

PM. The simulation model delivers accurate prediction results for PM-1 and PM-2 in both consolidation scenarios, as shown in Table 5. In light of the extent to which response times of these workloads increase in consolidation, the quality of results is impressive. Conversely, the product form model delivers larger errors and is not competitive except for the case of PM-1 in Con 2. The reason for the poor prediction of the product form model may be that requests of this workload type get merged in the VMM, which is not accounted for in the analytical equation. As Table 6 conveys our methodology estimates merging values ω of approximately 2.5 and 4.5 for Con 2 and Con 3 respectively. Larger ω's for Con 3 are reasonable, since more requests get queued at higher utilization levels resulting in more optimization operations by the VMM. As we have shown in Figures 6 (a) and (b) the amounts of requests larger than 512kB are very small for PM workloads, thus splitting operations are negligible. Figures 9 (a) and (b) show that predictions of the simulation model underestimate the measured response times for Con 2. We reason this might be due to the necessary approximation of service times for merged requests, where we estimate the aggregate service requirement with the mean. Even though the storage device likely serves merged requests asynchronously, this might be an optimistic approximation. Figures 10 (a) and (b) also show optimistic predictions for Con 3, with only a single exception for PM-2 submitted by VM 3. Conversely, the reference product form



(a) PM-1 (b) PM-2 (c) FFSB-2 Seq (d) FFSB-2 Rand

Figure 9: Comparison of Response Times from Measurement (Meas), Simulation (Sim), and Product-Form (Prod) Model for Two VMs Consolidation Scenarios With Largest Prediction Errors. Legend Shown in Figure (a) is Identical for Figures (b), (c), and (d).



(a) PM-1 (b) PM-2 (c) FFSB-1 Rand (d) FFSB-2 Rand

Figure 10: Comparison of Response Times from Measurement (Meas), Simulation (Sim), and Product-Form (Prod) Model for Three VMs Consolidation Scenarios With Largest Prediction Errors. Legend Shown in Figure (d) is Identical for Figures (a), (b) and (c).

model grossly overestimates response times across all VMs and consolidation scenarios.

FFSB. Both approaches deliver good response time predictions for Con 2, where the simulation model performs especially well for the cases of FFSB-1_S/R. Interestingly, the product form model works significantly better than for the PM workloads. We explain these good results due to the fact that FFSB requests are significantly larger in size, as shown in Figures 6 (c) and (d), and probably undergo less optimization operations in the VMM. Our technique performs VMM level splitting operations on the arrival trace, as well as merging, as shown Table 6. We have especially found the splitting behavior difficult to model, as it needs to maintain temporal dependence in the measured arrival trace. Furthermore, one needs to rejoin split requests and make additional approximations on the aggregate response time. For the case of Con 2 splitting and merging operations in our model almost compensate each other, supporting our assumption that net merging operations in the VMM are small in this scenario. Figures 9 (c) and (d) show response time values for the largest prediction errors and further illustrate that our model is optimistic. In Con 3 the simulation model performs extremely well for FFSB-1_S and FFSB-2_S. The product form solution delivers better results when workloads have random access patterns. Predictions are especially difficult for the simulation model in the case of FFSB-1_R. Here, Figure 9 (c) indicates predictions atypically are overestimating system measurements. While still being competitive, errors are also larger for FFSB-2_R. Figure 9 (d) confirms the earlier observation that our modeling assumptions lead to optimistic predictions.

Summary. Our findings indicate that prediction quality of disk request response times can be increased with enhanced models, which can account for splitting and merging operations of the disk scheduler. In cases where workloads do not allow large amounts of optimization operations product-form solutions can also be a valuable prediction tool.

6. RELATED WORK

A large amount of research literature is concerned with scheduling algorithms for disk IO in virtualized environments. The main challenges regarding scheduling are to provide fair access to the shared storage resource for all in-VM applications, while maintaining performance isolation, i.e. disk accesses by one application should not affect the IO performance of another. This work can be structured into approaches concerned with scheduling disk access on a single VMM [23] and methods that coordinate IO scheduling across multiple independent virtualized servers sharing a storage device [21].

Performance isolation in presence of resource contention is studied in [25]. The authors consolidate different types of workloads, i.e. CPU bound and disk bound, and derive mathematical models to predict relative performance compared to a normalized performance score. Degradation of end-to-end application performance due to server consolidation is investigated in [29]. Closer to our work, [9] derive a mathematical model to predict disk IO throughputs when moving from a native system to an isolated VMware ESX server environment. [22] measure disk workload characteristics and performance metrics in a consolidated virtualized environment. Contrary to us they do not consolidate by placing multiple workloads on the same LUN, but consolidate multiple LUNs into a single RAID group.

Queueing models are a popular tool to model the performance of shared resource environments [13]. One approach is to use queueing theory in order to predict performance attributes of applications when migrated from a native to

a virtualized environment [12]. A shared server environment is modeled as a time-domain queueing model with GPS scheduling in [18] in order to compute and assign resource shares. In [24], layered queueing networks are used to model multi-tier applications hosted in consolidated server environments. Recently, [26] propose an iterative model training technique based on artificial neural networks for dynamic resource allocation in consolidated virtualized environments. While some of the work above captures coarse grained disk requirements in the model in order to predict effects of resource allocation changes on performance of consolidated applications, none specifically tries to predict fine grained disk IO request performance degradation due to workload consolidation.

Prediction of disk request response time granularity based on a machine learning technique is presented in [30]. The approach employs Classification And Regression Tree (CART) models and treats the storage device as a blackbox. However, the model requires a training period and does not consider shared resource access.

7. CONCLUSIONS AND FUTURE WORK

We presented a trace-driven simulation model that can predict response times of disk IO requests when consolidating workloads on to a shared storage device. Our contributions are twofold. Firstly, we parametrize the model with in-VM measurement data only instead of instrumenting the VMM. Secondly, we introduce techniques that quantify and incorporate splitting and merging operations inherent to a VMM level disk scheduler. The proposed methodology is validated against system measurements and produces better results than an established product form solution for certain homogeneous workload types. For future work we plan to validate the proposed methodology on other virtualization platforms with similar system characteristics. We also intend to extend the methodology for mixed workloads, which will require improvement of our search technique to determine individual model merge parameters depending on workload type. In order to predict end-to-end performance degradation due to storage contention effects, we need to investigate how disk request performance at the VM operating system level correlates to application performance.

8. ACKNOWLEDGEMENT

This research has been partially funded by the InvestNI/SAP VIRTEX project. The work of Giuliano Casale has been funded by the Imperial College Junior Research Fellowship Scheme. Thanks to Stephen Dawson for valuable comments.

9. REFERENCES

[1] blktrace-linux man page. `http://linux.die.net/man/8/blktrace`.

[2] FFSB-v6.0-rc2. `http://sourceforge.net/projects/ffsb`.

[3] openfiler. `http://www.openfiler.com`.

[4] Postmark-1.51-7. `http://packages.debian.org/sid/postmark`.

[5] RHEL 5 IO tuning guide. `http://www.redhat.com/docs/wp/performancetuning/iotuning/index.html`.

[6] Storage queues and performance. `http://communities.vmware.com/docs/DOC-6490`.

[7] vmware. `http://www.vmware.com`.

[8] I. Ahmad. Easy and efficient disk I/O workload characterization in VMware ESX server. In *IISWC*, pages 149–158. IEEE, 2007.

[9] I. Ahmad, J. M. Anderson, A. M. Holler, R. Kambo, and V. Makhija. An analysis of disk performance in VMware ESX server virtual machines. In *WWC-6*, pages 65–76. IEEE, 2003.

[10] J. Axboe. Linux block IO - present and future. In *Linux Symposium*, pages 51–61, 2004.

[11] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22(2):248–260, 1975.

[12] F. Benevenuto, C. Fernandes, M. Santos, V. Almeida, J. Almeida, G. Janakiraman, and J. Santos. Performance models for virtualized applications. In *ISPA*, volume 4331 of *LNCS*, pages 427–439. 2006.

[13] M. N. Bennani and D. A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC*, pages 229–240. IEEE, 2005.

[14] D. Boutcher and A. Chandra. Does virtualization make disk scheduling passé? *SIGOPS OSR*, 44(1):20–24, 2010.

[15] A. D. Brunelle. blktrace user guide, 2007.

[16] G. Casale, N. Mi, and E. Smirni. Bound analysis of closed queueing networks with workload burstiness. In *SIGMETRICS*, pages 13–24. ACM, 2008.

[17] G. Casale, E. Z. Zhang, and E. Smirni. KPC-toolbox: Simple yet effective trace fitting using markovian arrival processes. In *QEST*, pages 83–92, 2008.

[18] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *IWQoS*, pages 381–398. Springer, 2003.

[19] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM*, pages 1–12. ACM, 1989.

[20] P. Goyal, H. M. Vin, and H. Chen. Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks. In *SIGCOMM*, pages 157–168. ACM, 1996.

[21] A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional allocation of resources for distributed storage access. In *FAST*, pages 85–98. USENIX, 2009.

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

[23] W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. *ACM PEVA*, 32(1):37–48, 2004.

[24] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *ICAC*, pages 23–32, 2008.

[25] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu. An analysis of performance interference effects in virtual environments. In *ISPASS*, pages 200–209. IEEE, 2007.

[26] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application performance modeling in a virtualized environment. In *HPCA*, pages 1–10, 2010.

[27] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.

[28] N. Mi, Q. Zhang, A. Riska, E. Smirni, and E. Riedel. Performance impacts of autocorrelated flows in multi-tiered systems. *Elsevier PEVA*, 64(9-12):1082–1101, 2007.

[29] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin. Performance evaluation of virtualization technologies for server consolidation. Technical Report HPL-2007-59, HP Laboratories Palo Alto, 2007.

[30] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger. Storage device performance prediction with cart models. In *MASCOTS*, pages 588–595. IEEE, 2004.