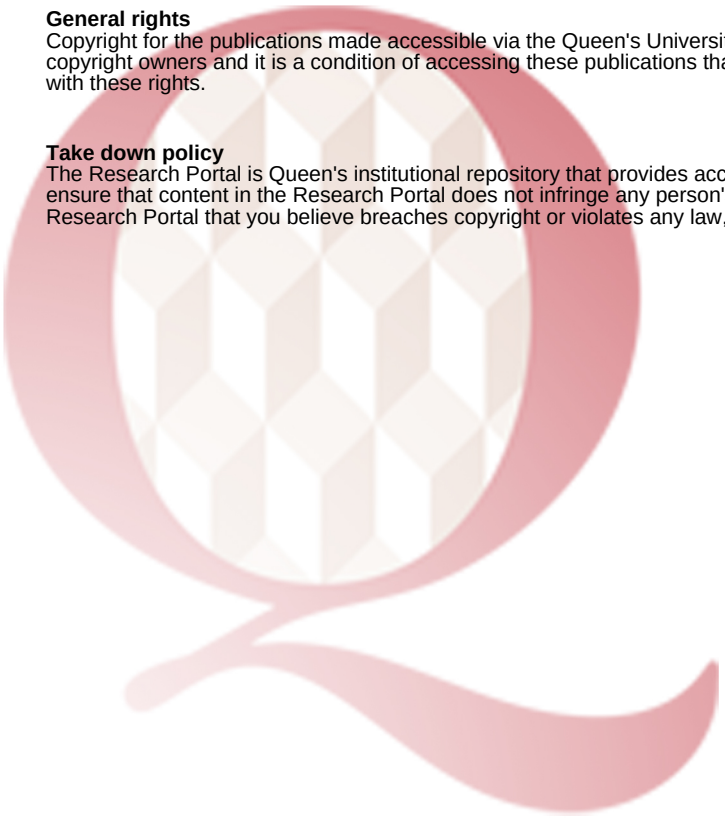
Performance models of storage contention in cloud environments

Kraft, S., Casale, G., Krishnamurthy, D., Greer, D., & Kilpatrick, P. (2013). Performance models of storage contention in cloud environments. Software and Systems Modeling, 12(4), 681-704. DOI: 10.1007/s10270-012-0227-2

Published in:
Software and Systems Modeling

Document Version:
Early version, also known as pre-print



Noname manuscript No.
(will be inserted by the editor)

Performance Models of Storage Contention in Cloud Environments

Stephan Kraft · Giuliano Casale · Diwakar Krishnamurthy · Des Greer · Peter Kilpatrick

Received: date / Revised version: date

Abstract We propose simple models to predict the performance degradation of disk requests due to storage device contention in consolidated virtualized environments. 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 server. We first propose a trace-driven approach that evaluates a queueing network with fair share scheduling using simulation. The model parameters consider Virtual Machine Monitor (VMM) level disk access optimizations and rely on a calibration technique. We further present a measurement-based approach that allows a distinct characterization of read/write performance attributes. In particular, we define simple linear prediction models for I/O request mean response times, throughputs and read/write mixes, as well as a simulation model for predicting response time distributions. We found our models to be effective in predicting such quantities across a range of synthetic and emulated application workloads.

Stephan Kraft
SAP Research, Belfast, UK, E-mail: stephan.kraft@sap.com

Giuliano Casale
Imperial College London, Dept. of Computing, London, UK, E-mail: g.casale@imperial.ac.uk

Diwakar Krishnamurthy
University of Calgary, Dept. of ECE, Calgary, AB, Canada, E-mail: dkrishna@ucalgary.ca

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

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

Keywords Performance Modeling · Virtualization · Storage

1 Introduction

The performance of I/O-bound applications is dominated by the time required by the operating system to schedule read and write operations and by the response times of the storage devices in completing such requests. Since changes in the workload, as well as in the software and hardware environments, can affect the latency of disk I/O requests, it is often useful to define performance models to anticipate the effects of a change. This is especially important in Cloud environments that employ virtualization technologies. In virtualized data centers 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 [34]. In such systems estimates of I/O contention for a given VM placement configuration can support management and consolidation decisions.

However, modeling the performance of disk requests is very challenging due to the joint interaction of I/O flows issued by several VMs and because of the complexity of caching mechanisms, scheduling algorithms, device drivers, and communication protocols employed by both the VMs and the VMM. Read and write requests may affect system performance in very different ways. Ganger and Patt [27] introduce three classes of requests based on how individual request response times influence system performance. An I/O request is considered *time-critical* if the generating thread blocks until the request is completed, e.g. a process is halted until a synchronous read request has been completed. *Time-*

limited requests must be completed within a given amount of time otherwise they will become time-critical, for example file system read-aheads. Requests that do not require waiting times of the submitting process are classified as *time-noncritical*. Such disk I/O requests must be completed to maintain stable copies of non-volatile storage, for example background flushes of asynchronous writes. Time-noncritical requests can indirectly impact performance when interfering with the completion of more critical requests, thus model definition is complicated by the interaction between these different workload behaviors.

In this paper we tackle this complexity by introducing simple models for I/O performance prediction in consolidated environments where multiple VMs can share access to a remote storage server. Our methodology, summarized in Figure 1, requires first to study VMs when they run in isolation on a virtualized server. Based on collected measurements we propose two types of models to forecast the impact of consolidation on I/O performance.

Specifically, for each VM of interest we collect traces of arrival times, estimated service times, and arrival queue lengths for I/O requests. We develop two modeling techniques that embody this approach. We start by developing what we denote as the Homogeneous model to handle scenarios where multiple VMs running the same type of workload are consolidated. We then extend this technique by developing what we denote as the Decomposition model. In addition to handling scenarios where consolidated VMs run heterogeneous workloads, this technique supports distinction between read and write requests.

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

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

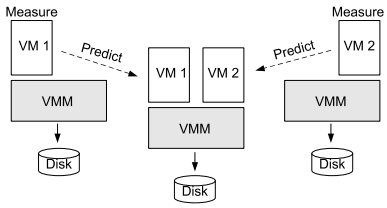


Fig. 1 Problem Approach.

Extensive experimentation on test workloads generated with the Postmark (PM), FFSB, and Filebench disk benchmarks reveals that our methodologies can forecast successfully consolidation effects on I/O performance. Summarizing, the main contributions of this paper in predicting consolidated workload performance are threefold.

1. Our methodology allows us to parameterize models based on data obtained inside VMs in isolation experiments. It requires very little information from the VMM thereby effectively treating the VMM as a blackbox. It also obviates the need to collect model training data for different VM consolidation scenarios.
2. The trace-driven simulation model is enhanced with an iterative calibration technique that approximates parameterization inaccuracies in the absence of detailed VMM level measurements.
3. The measurement-based decomposition model defines a set of simple analytical estimators to approximate performance of read/write disk requests separately.

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 Homogeneous modeling methodology is presented in Section 4 and its validation results are shown in Section 5. A refined model using the decomposition approach is illustrated in Section 6, while Section 7 presents the corresponding validation results. Section 8 gives an overview of related work. Section 9 offers summary and conclusions.

2 Motivational Example

In consolidation scenarios where more than a single VM submits large numbers of I/O requests, competition for the disk drive can lead to significant increases in latencies, i.e. response times. To illustrate the problem Figure 2 compares the mean response times in three experiments conducted on our reference system, which is introduced in Section 3.1. In the first two experiments,

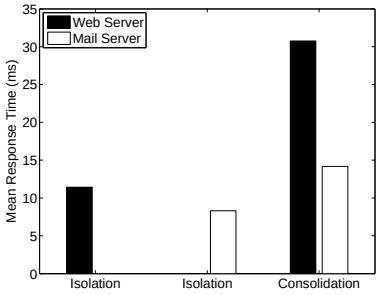
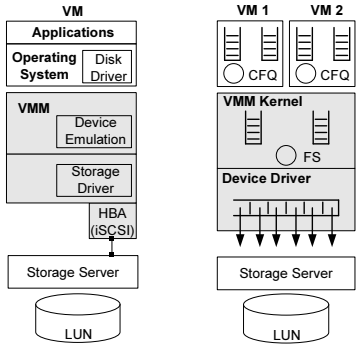


Fig. 2 Effect of VM Consolidation on Disk Request Latencies.



(a) Storage Subsystem (b) Storage Queues

Fig. 3 I/O Architecture and Storage Queues of the Reference System.

denoted isolation experiments, single VMs run on the system thus avoiding contention from other VMs. In the third experiment, denoted consolidation experiment, we run two VMs on the same virtualized server and find that their disk requests experience storage contention. Response times are computed as the time between request issue and request completion as recorded by the VM operating system. In consolidation we record large mean response time increases of roughly 170% and 71% for web and mail server type disk workloads, respectively. This makes a strong case for the significant performance impact that consolidation can have on end-to-end response time of I/O requests and motivates the investigation in this paper. Specifically, the example motivates the need for accurate models that can capture and quantify the possible I/O 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 tools used to obtain I/O measurements. Our modeling techniques have been tested only on the architecture described below, but we believe them to be representative of virtualized environments 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 [4] storage server via the iSCSI protocol and 1 GBit Ethernet. The storage server comprises 8 Intel Xeon processors clocked at 2GHz and 15GB of RAM. It manages a 3ware 9000 series SATA-II hardware RAID controller with 256MB of RAM. The 15 disc RAID 5 array uses a stripe size of 64k and disk write-caching is turned off.

On the server we host the virtualization platform VMware ESX Server 3i - 3.5.0 [9], which accesses the storage device through a software iSCSI host bus adapter (HBA). We specifically choose VMware, as it is a widely diffused virtualization technology. The virtualized environment consists of multiple Debian 5.0.1, kernel 2.6.26-2, guest VM systems, each with identical configuration of 1 virtual CPU, 500MB RAM, and 50GB of "virtual" hard disk formatted as *ext3* file system.

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

The VMM formats and stores virtual disk files in the Virtual Machine File System (VMFS). The VMFS provides distributed locking mechanisms facilitating concurrent access to virtual machine disk files from multiple physical machines. A VMFS volume may also be extended over multiple LUNs for storage pooling [3].

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

Disk I/O requests issued from a VM consist of one or multiple contiguous blocks for either reads or writes. Once an application running inside the VM submits a request, the request first goes to the disk driver of the VM operating system as shown in Figure 3 (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 [13].

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 multiple optimizations such as aggregating and reordering operations by in-VM schedulers, as well as request splitting at the VMM level. Such operations are used to optimize disk access patterns, e.g., by aggregating multiple requests for small amounts of contiguous blocks to fewer requests for large amounts of contiguous blocks. In virtualized environments these operations can have a significant impact on disk request performance, as scheduling policies at VM and VMM level may impair each other [20].

In our environment the Debian guest VMs are configured to use the completely fair queueing (CFQ) [14] scheduler, which has per default 64 internal queues to maintain and keep disk I/O requests [6]. 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 [30], controlled with a fair-share (FS) [25] scheduler. This class of algorithms schedules shared resources among competing flows of request classes where each request is associated with a cost [28,29,19]. Resource sharing entails the need to control how consolidated request classes consume the resource in order to isolate performance and manage differentiated resource access. Without such management load surges of competing flows may impair each other unacceptably. FS allocates resource capacities in proportion to weights that have been assigned to the competing request classes. See Section 4.1 for more detailed information on the scheduling.

Furthermore, the server maintains a device driver queue for each LUN, which controls the *issue queue length* defined as the number of pending requests the server can have at the storage device at a time [7]. Virtualized servers typically allow configuration of 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 [15] is complicated due to complex splitting, aggregating, and reordering operations on arriving requests that depend on spatial locality on the storage device. Although forking and joining operations may alleviate such difficulties, they are hard to parameterize in the absence of direct measurement of the VMM internal optimization, which is the typical situation in practice. Furthermore, the batched submission of requests can result in 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 [41,22] and we handle them in this work by resorting to trace-driven simulation.

3.2 Measurement Tool

This section describes the monitoring tools we have used to collect disk I/O traces in order to quantify the performance of disk requests. Traces are captured at two system layers: the virtualization and the storage server. Our traces comprise a 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. We note that we calculate per request response times as the time difference between completion and issue events in the trace. We are aware of the timekeeping inaccuracies in virtualized environments and use the network time protocol (NTP) to synchronize time between VMs. NTP has been reported to work fairly well on our reference system [10].

Measurements on the virtualized server are obtained inside VMs with the block layer I/O 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. In order to monitor the I/O request trace submitted from the VMM to the storage server, we use the network protocol analyzer *tshark* [8] to intercept iSCSI network packets. The tool is installed at the storage server and only collects the headers of iSCSI packets, which enables us to extract operational codes and control information without handling iSCSI payloads.

4 Homogeneous Model

Our model comprises trace-driven simulations to predict the performance degradation of VM disk request

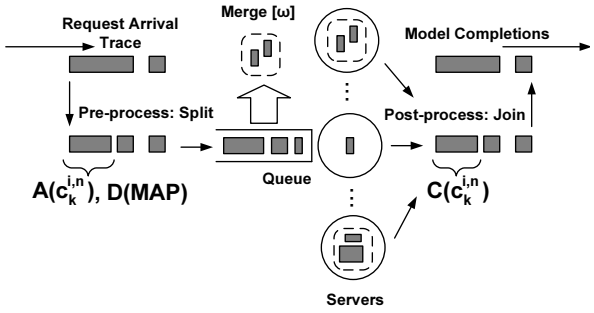


Fig. 4 Methodology Overview.

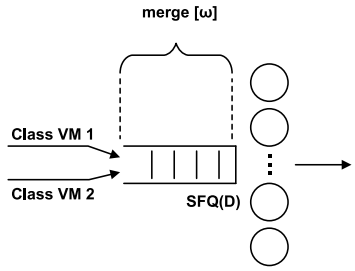


Fig. 5 Queueing Model.

response times due to storage device contention in homogeneous workload consolidations, i.e. scenarios where multiple workloads of the same type are submitted from a single server. The fundamental idea of our approach is to first record application traces in isolation benchmark experiments and then utilize these traces to parameterize simulation models of consolidation scenarios. The challenge is to define realistic models of contention and of the internal VMM I/O optimizations which affect the end-to-end delays of read and write operations. Our methodology considers a heuristic for request splitting behavior at VMM level and introduces a model calibration technique to alleviate parameterization inaccuracies, e.g. on request service times. Model parameterization is solely based on measurements obtained within the VMs and information that we gathered from available documentation on VMM operations such as fair-share scheduling and splitting. 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, where each i^{th} arriving request of class k, denoted c_k^i, may be split into n requests. Section 4.2.1 explains in detail how request arrival times $A(c_k^i)$ are assigned. Requests are 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) [32] scheduling policy across a pool of servers, each representing a parallel connection to the LUN. This model additionally introduces a calibration technique by means of a merging heuristic. The merging bundles a configurable number of ω requests and enables them to share a server. The queueing network and the scheduling algorithm are presented in 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 $C(c_k^i)$. 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.

The model implementation extends JINQS [26], a library for simulating multiclass queueing networks. Based on available documentation on the VMM, we implemented a practical SFQ disk scheduling discipline to schedule requests on to the servers. Fair queueing [25] 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) [32], a special variation of SFQ [29], which has previously been used in related work [30] to model our reference system. The depth parameter D controls the number of concurrent requests in service and consequently corresponds to the number

Table 1 Summary of Input and Output Parameters for the Homogeneous Model.

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

of servers in our model. Upon arrival of request c_k^i, it is assigned a *start tag* $S(c_k^i)$ and a *finish tag* $F(c_k^i)$ by the scheduler. The tag values represent the times at which each request should start and complete service according to a system notion of *virtual time* $v(t)$. Tags are computed as:

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

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

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 scheduling algorithm reserves the specified minimal share to each class. In cases where a class has no active requests surplus resources are shared among active classes according to their relative weights.

The scheduler issues a maximum of 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 requests. In this step, we account for the request splitting operations of the VMM which are triggered when the block sizes of arriving requests exceed a certain threshold.

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:

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

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

$$B(c_k^{i,n}) = \begin{cases} l_{max} & n \in \{1..N_k^i - 1\} \vee \\ & B(c_k^i) \bmod l_{max} = 0 \\ B(c_k^i) \bmod l_{max} & n = N_k^i \wedge \\ & B(c_k^i) \bmod l_{max} \neq 0, \end{cases} \tag{5}$$

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 $C(c_k^i)$. We compute the response times of joined requests as the mean:

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

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 for specifying queueing models and are typically estimated based on direct data measurement and statistical inference. A common approach to characterizing the resource consumption of requests is to monitor system utilization and use regression techniques based on operational laws [38]. 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.

Request service times belonging to a specific VM collected during isolation experiments are fitted to a MAP [23], 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 Model Calibration Methodology

Our simulation experiments indicate that prediction results can be significantly improved if the model is calibrated with a merging heuristic to account for inaccuracies in parameter approximations. Calibration methods can aid in building simple models of complex systems [16], e.g., in case studies like ours where detailed documentation of commercial software internals are not available. As we mentioned previously, we allow the scheduler to merge queued requests and use the merging to compensate for possible deficiencies in the parameterization of our model. For example, deficiencies in service times may arise due to the burstiness inherent in disk workloads and the resulting impact on storage system behavior [43]. By enabling the scheduler to merge, i.e. aggregate, queued request we effectively model such batching behaviour. Furthermore, our analysis of storage server monitoring data in Section 6.3.1 shows evidence that, under simplifying assumptions, service times at the disk can be load dependent. For example, service times can decrease at high loads. Since we approximate service times from low load measurements, the merging of requests may help modeling of such load dependent behaviour. The merging algorithm is described in Section 4.3.1. The iterative technique to quantify the amount of scheduler merging operations performed for a given workload configuration is described in Section 4.3.2.

4.3.1 Request Merging Algorithm

The model merges a configurable number of queued requests in such a way that they still remain separate entities, but share a server, i.e., connection to the LUN. As shown in Algorithm 1, we pass a merge value parameter, denoted ω, to the simulator, which serves as a modeling abstraction in our blackbox model. Merging values can range in $[1, \infty]$, where we consider a merge value of 1 as a single job being issued per service station, i.e. no merging, whereas a large ω indicates that several requests are merged together before issue to the server. Since requests are indivisible, we implement a function *get_int_value* to obtain from ω the number of merged requests in each round of calibration. For example, if ω is configured as 2.5 there is an equal probability that the maximum 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

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
```

is of a different class as the already merged requests. Once a merged job has received service and exits the model, each of the merged requests counts as a separate completion. Since each of the requests sharing a service station has an individual service time, we approximate the aggregate service requirement of merged requests with the mean of the individual request service times.

4.3.2 Merge Value Estimation

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

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.

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

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

To validate this linear assumption Table 2 shows measurements of the mean number of requests from benchmark experiments in our reference system. We present measurements for a number of workload configurations averaged over multiple benchmark runs. For detailed information on the considered workloads see Section 5.1. Results indicate that the linear assumption is a good approximation of system behavior. We expect our assumption to hold as long as the aggregate mean number of requests outstanding from VMs, i.e. requests issued and not yet completed, does not exceed the number of available connections from the VMM to the LUN.

Iterative Search. The expected queue length approximation is an input parameter for an iterative search technique, which we propose to estimate the merge value parameter for the simulation model. As shown in Algorithm 2, the search is further parameterized with a configurable initialization point and a maximum relative error value, Δ_{max}, that serves as a search termination condition. Each search iteration begins with a number of simulator runs that incorporate merging operations according to the current value of ω. Every simulator run is parameterized with a random combination of inter-arrival 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 esti-

Algorithm 2 Iterative Estimation of Merge Value

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

mate according to the error function

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

where N^K_{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^K_{exp}. In cases where the estimation error is outside this range, we control search direction and ω values on the basis of a binary search. Let $\omega = g(N'_{sim})$ be the merge value used in simulation to obtain N'_{sim}. If N'_{sim} is smaller than N^K_{exp}, we decrease ω in order to increase the mean number of requests in simulation. In cases where previous iterations have produced a $N'_{sim,old}$, such that $\{N'_{sim,old} > N^K_{exp} > N'_{sim}\}$ the merge value for the next iteration is determined by

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

which is half the distance between ω values used to obtain 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. Table 1 offers a summary of the input and output parameters for the Homogeneous model.

Table 3 Workload Configurations.

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

5 Validation Experiments: Homogeneous 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) [5] is a synthetic single-threaded workload comprising small file and metadata-intensive operations designed to emulate I/O patterns of Internet applications such as e-mail and e-commerce. We specifically choose these two workload types to validate our model since the different file size distributions should result in distinct quantities of VMM I/O request splitting operations.

In order to obtain a system steady state, benchmarks for each configuration are submitted over a period of 75 minutes in five minute intervals. We have considered two configurations of a PM workload in our investigation, denoted PM-1 and PM-2. The workloads differ in the file sizes of the initial file set, as well as the sizes of read and write requests (see Table 3). The "size" parameters specify how much data is read or written to files at a time.

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 [6]. 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. More detail on the considered workloads can be found in previous work [35].

5.2 Workload Characterization

Request Size. The disk I/O scheduler of the VM operating system aggregates and reorders queued requests. As a result the total number and sizes of disk requests issued by the VM operating system to the VMM can significantly differ from the total number and sizes of requests submitted by the benchmark application to the VM operating system. To illustrate how workloads submitted by the considered application configurations are translated into logical block requests by the VM operating system, 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 [12].

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). For both FFSB workloads large proportions of requests are merged to block sizes > 128, which corresponds to request sizes $> 512kB$.

Interarrival Times. The significantly different characteristics of the considered workload configurations are also reflected in the interarrival times. Table 4 shows that mean interarrival times at the VMM are roughly four times larger in the case of FFSB workloads compared to the PM configurations.

Since our model uses information recorded during isolation benchmark experiments only, we are especially interested in quantifying the impact of workload consolidation on request interarrival times. Figure 7 shows the interarrival time distributions of disk requests submitted from VMs when running in isolation (Iso) compared to consolidation scenarios with additional VMs submitting an identical workload in the background. Interestingly, none of the arrival distributions from the VM to the VMM displays a large deviation from their corresponding isolation distributions when the workload on the server is doubled (Con 2) or tripled (Con 3). We take this as a clear indication that queueing of disk I/O requests due to resource contention takes place at the VMM layer, rather than at the VMs themselves.

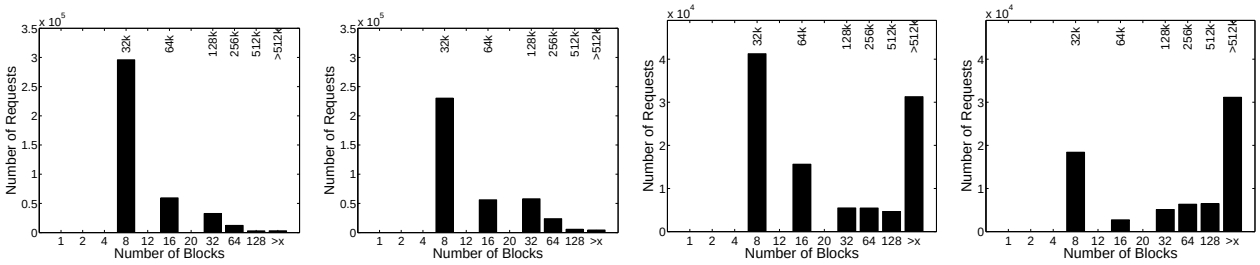
Table 4 Mean in ms, Standard Deviation (std), and Coefficient of Variation (cv) Statistics for Request Interarrival and Service Times.

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

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

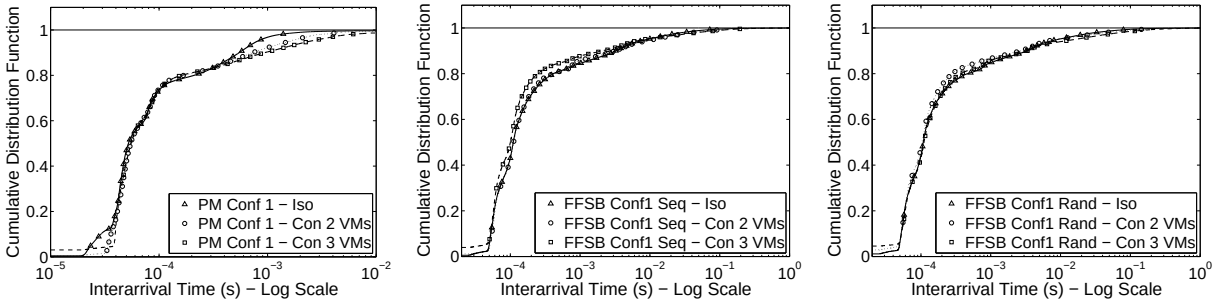
5.3 Model Validation

For validation we conduct a series of benchmark experiments on our reference system using the previously introduced workload configurations. Workloads are submitted from within VMs. We consider scenarios with up to three VMs, denoted VM 1, VM 2, and VM 3, where we consolidate homogeneous workloads, i.e., workloads of the same type. We specifically choose this number of VMs as it resembles a realistic situation for the consolidation of disk I/O intensive applications and the storage contention scenario in our reference system [3]. Each benchmark experiment consists of 15 individual 5min runs and results are reported as means over all



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

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



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

Fig. 7 Impact of Workload Consolidation on Arrival Processes of PM and FFSB Workloads.

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 5 shows that workload consolidation leads to an increase in disk I/O response times across all workload configurations. The PM workloads suffer a higher performance degradation than FFSB, with an increase ranging in approximately [158%; 199%] and [299%; 468%] in the two and three VM consolidation scenarios, respectively. In case of the FFSB workloads this increase is less severe, but still ranges approximately in [25%; 42%] and [47%; 89%] over all VMs and configurations. Furthermore, there is no clear trend showing that response times of random FFSB workloads increase to a larger degree than sequential FFSB workloads when consolidating. Interestingly, VMs seem to have slightly different I/O characteristics even for identical workload configurations. For example, in a three VM consolidation scenario PM-2 requests submitted from VM 2 have a mean response time of 56ms, while the mean response time of PM-2 requests submitted from VM 3 is 43.9ms. Such discrepancies may be explained due to the spatial locality of VM disk images on the LUN. We indicate the stability of our monitoring using a 95% confidence interval and report the confidence interval width CI_{iso} as the percentage of the mean measured response time in isolation over all benchmark runs.

5.3.2 Prediction Accuracy

We validate model predictions of mean disk request response times 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 [18]. Response time estimates for the product form model are determined analytically as

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

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

Table 5 Mean Response Time Measurements of Disk I/O Requests in ms for VMs in Isolation (Iso), Confidence Interval Width of Isolation Measurements (CI_{iso}), and Mean Response Time Measurements of Consolidation Scenarios with Two VMs (Con 2) and Three VMs (Con 3).

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

Table 6 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 7 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

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 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

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

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

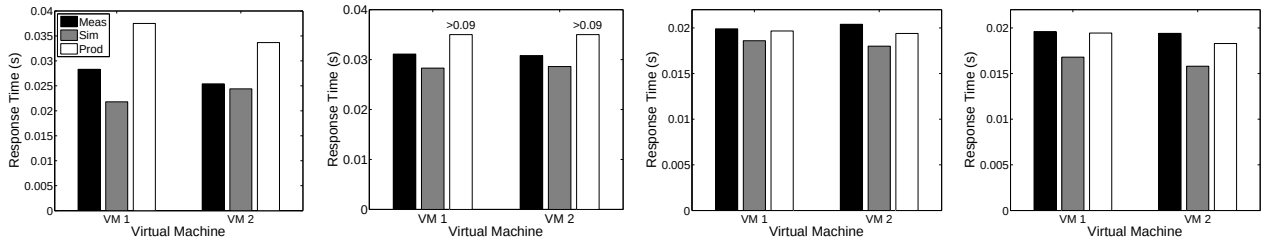
$$\Delta = \sum_{k=1}^{K} \frac{1}{K} \left| \frac{C_{meas,k} - C_{est,k}}{C_{meas,k}} \right|, \tag{13}$$

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

PM. The simulation model delivers accurate prediction results for PM-1 and PM-2 in both consolidation scenarios, as shown in Table 6. In light of the extent to which response times of these workloads increase in consolidation, the quality of results is highly satisfactory. Conversely, the product form model delivers larger errors and is not competitive except for the case of PM-1 in Con 2. This result illustrates the effectiveness of the model calibration technique, as Table 7 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 merging opportunities by the simulation model.

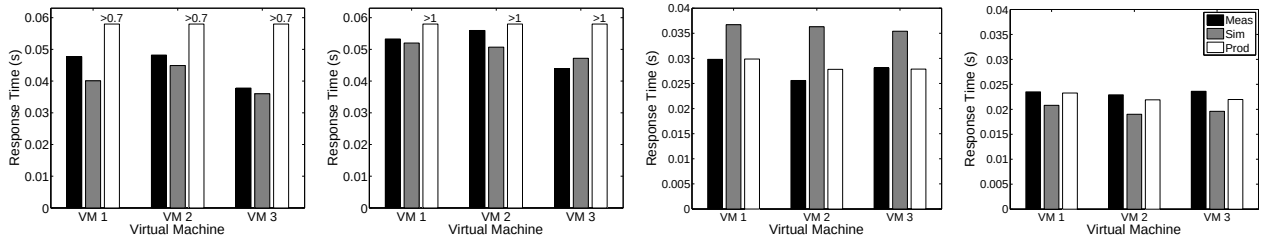
As we have shown in Figures 6 (a) and (b) the numbers of requests larger than 512kB are very small for PM workloads, thus splitting operations are negligible. Figures 8 (a) and (b) show that predictions of the simulation model underestimate the measured response times for Con 2. We reason this might 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 9 (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 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. Our technique per-



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

Fig. 8 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

Fig. 9 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).

forms VMM level splitting operations on the arrival trace, as well as merging, as shown Table 7. In particular we have 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. Figures 8 (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 that 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 in homogeneous consolidation scenarios can be increased with enhanced models, which can account for splitting operations of the VMM disk scheduler and are heuristically calibrated. However, the range of application for such models significantly widens if they are also able to accurately predict response times in heterogeneous consolidation scenarios. Model calibration with multiple workload types is more difficult since each type requires individual ω merge values and thus complicates the merge value estimation step. We leverage our findings from modeling homogeneous workload consolidations and tackle this additional complexity with a model refinement as described in Section 6.

6 Decomposition Model

In this section we advance our modeling effort and propose models for predicting heterogeneous workload consolidations. Such models are more powerful as homogeneous ones, because they enable system administrators to find the workload mixes with the least interference effects. Specifically, we propose linear estimation formulas to forecast mean read/write mixes, throughputs, and response times in consolidation. Additionally, we introduce a simulation model for predicting response time distributions. Such a capability is particularly useful for cloud operators interested in providing Service Level Agreements (SLAs) to customers.

The refined models specifically distinguish between read and write request types. Such distinction is advantageous, as the performance of synchronous disk I/O read requests may directly affect application performance in cases where processes block until completion

of a previously submitted request. Write requests can usually be served asynchronously and do not require the submitting process to block. The differentiation according to request type allows us to decompose the workload and model read and write requests separately.

6.1 Static Analysis of Arrival Queue Lengths

The I/O workload of the storage device consists of a mixture of read and write requests. Due to the mixed nature of consolidated workloads, ideally the system must minimize situations where time-noncritical write requests may interfere with time-critical reads. We use the *tshark* tool to observe how the arrival queue length at the storage server is partitioned between read and write requests in one of our consolidation experiments, in order to study how our reference system submits requests of different types. Figure 10 (a) shows that a majority of arriving read requests only find reads ahead of them. This suggests that the VMM assigns a form of priority shares for read requests, e.g. similar to original UNIX systems (System 7) that used a disk request scheduler giving read requests non-preemptive priority. Mostly the utilization of the storage server is low, since only a small fraction of the 32 available connections from the VMM to the storage server are being used. However, there is a significant number of instances where the maximum amount of available connections are in use indicating high utilization periods. Interestingly, we also see a noticeable number of cases where the arrival queue contained 16 writes.

Figure 10 (b) shows more variability in the arrival queue lengths for write requests. On most occasions arriving write requests find a small number of reads, as well as other write requests roughly ranging in $[1; 16]$. These observations suggest that a maximum of 16 write requests are batched from the VMM to the storage device. However, there also are noticeable frequencies where arriving write requests see large queues of read requests. These cases could indicate situations where batches of write and read requests are submitted within a short time interval. We refer to previous work [21] for analysis of the dynamic behavior of arrival queues and a graphical illustration of I/O request batching.

6.2 Linear Predictor Formulas

Our performance prediction methodology considers the following performance metrics. We include definitions for two VM consolidation scenarios. The generalization is obvious and not given due to space limitations:

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

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

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

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

6.2.1 Approximation of Consolidation Read/Write Mixes

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

$$\beta_{i,j}^R \approx \alpha_i^R \left(\frac{T_i}{T_i + T_j} \right) + \alpha_j^R \left(\frac{T_j}{T_i + T_j} \right) \tag{14}$$

$$\beta_{i,j}^W \approx \alpha_i^W \left(\frac{T_i}{T_i + T_j} \right) + \alpha_j^W \left(\frac{T_j}{T_i + T_j} \right) \tag{15}$$

A mathematical justification of the above approximation is given in Appendix A.

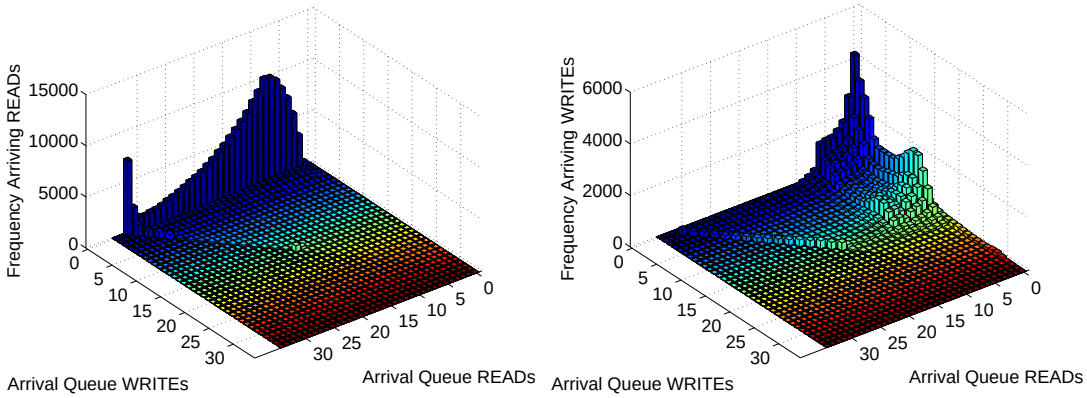
6.2.2 Approximation of Consolidation Throughputs

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

$$\begin{aligned} X_{i,j}^R &\approx T_i^R \left(\frac{T_i}{T_i + T_j} \right) + T_j^R \left(\frac{T_j}{T_i + T_j} \right) \\ X_{i,j}^W &\approx T_i^W \left(\frac{T_i}{T_i + T_j} \right) + T_j^W \left(\frac{T_j}{T_i + T_j} \right) \end{aligned} \tag{16}$$

Table 8 Summary of Input and Output Parameters for the Decomposition Model Linear Estimators When Predicting Read Requests in Two VM Consolidation Scenarios. The Notation for Write Requests is Similar.

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



(a) Arrival Queue as Seen by Arriving Reads (191786 Arrivals)

(b) Arrival Queue as Seen by Arriving Writes (152899 Arrivals)

Fig. 10 Arrival Queue Lengths From a Two VM Consolidation Experiment (Web+Mail).

6.2.3 Approximation of Consolidation Response Times

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

$$S_i^R \approx C_i^R/(1 + A_i^R), \tag{17}$$

where $1 + A_i^R$ is the queue-length seen upon arrival by a read request at ESX including the arriving job itself [36]. Equivalently, the arrival queue-length can be considered at the storage server if ESX data is not directly available, since we found that in isolation the difference between the two measurements is often negligible. Then we estimate the expected response time for a read request in consolidation as

$$C_{i,j}^R \approx C_i^R + S_j^R A_j^R \tag{18}$$

This approximation assumes that the overheads for reads in consolidation for VM i are only due to the interference with reads issued by the other VM j; write requests do not interfere with the response times of reads for VM i. Figure 10(a) supports this assumption, as the large majority of read arrivals only find requests of the same type in the system.

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

$$C_{i,j}^W \approx C_i^W + S_j^W A_j^W \tag{19}$$

Finally, similar expressions are defined for the response times of VM j, i.e., $C_{j,i}^R$ and $C_{j,i}^W$. We refer to Table 8 for a summary of parameters and estimators of the linear predictor formulas.

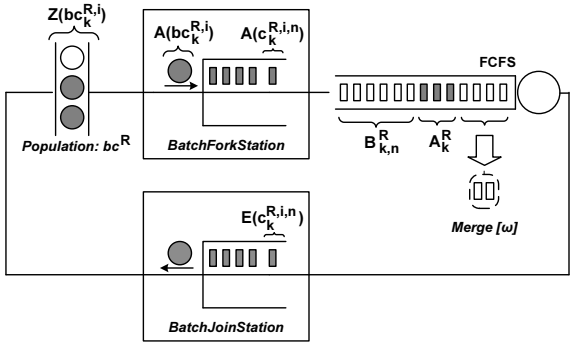


Fig. 11 Customized Closed Queueing Model.

6.3 Simulation Model

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

Our model follows the concepts of the linear estimators introduced in Section 6.2 and focuses on the mostly synchronous and performance critical read requests [44]. To capture the feedback effects between read request completions and arrivals we resort to a multiclass closed queueing model as sketched in Figure 11. Requests submitted from individual VMs are distinguished in K separate classes. Based on the assumptions and approximations in Section 6.2.3 the simulation model uses a single server with FCFS scheduling.

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

$$(n \in \mathbb{N}, 1 \leq n \leq b^R) : A(c_k^{R,i,n}) = A(bc_k^{R,i}), \tag{20}$$

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

Algorithm 3 Parameterization of Simulation Model

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

6.3.1 Model Parameterization

The parameterization process of the simulation model consists of two steps as outlined in Algorithm 3. First we use measurements obtained in a single representative isolation benchmark experiment to determine disk I/O request batch sizes, batch think times, and service times. Secondly, we find an approximation of the model population by modulating this parameter until throughputs in simulation match a throughput estimate computed with the linear predictor formulas. See the following paragraphs for detailed definition of model parameters.

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

$$\forall c_k^{R,i} \in B_{k,n}^R : \text{diff}\{A(c_k^{R,i+1}), A(c_k^{R,i})\} \leq t, \tag{21}$$

where batch $B_{k,n}^R$ holds all read arrivals $c_k^{R,i}$ that have interarrival times ranging within a timeout constraint t. We specify t as $2ms$, an approximation based on the analysis of the dynamic arrival queue lengths as described in [21]. Similar quantities $c_k^{W,i}$, $B_{k,n}^W$, $A(c_k^{W,i})$ are defined for write requests.

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

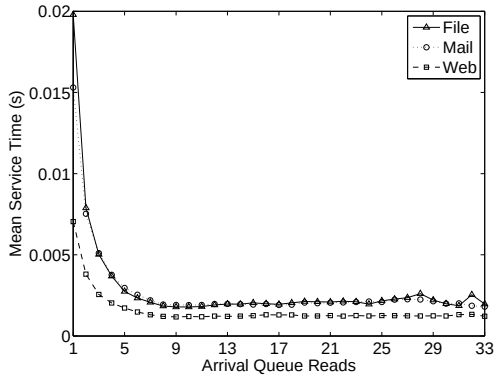


Fig. 12 Measured Load Dependent Service Times of Reads for File, Mail, and Web Server Workloads in Isolation.

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

$$Z^R_{k,n} = \text{diff}\{\min(A(c^{R,i}_k), c^{R,i}_k \in B^R_{k,n+1}), \max(E(c^{R,i}_k), c^{R,i}_k \in B^R_{k,n})\}, \quad (22)$$

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

Service Times. We approximate the service requirement of requests using the well-known mean value analysis (MVA) expression introduced in (17). Our workload characterization study has shown that, under the simplifying assumption of FCFS scheduling, service times during phases of lower loads are larger than during phases of higher loads. We explain this behavior due to the disk device's capabilities to serve large batches of requests faster than small numbers of individual requests, i.e. the per request service times of a batched sequential read versus service requirements of multiple random reads. Figure 12 reveals that read request service requirements of all considered workload types are load dependent. We estimate the mean request service time as a function of the queue seen on arrival by

$$D(c^R_{k,a_k}) \approx \frac{1}{n}\sum_{i=1}^{n} C(c^{R,i}_k)/(1+A^R_k), a_k = (1+A^R_k), \quad (23)$$

where $D(c^R_{k,a_k})$ is the mean service time of read requests given the arrival instant queue length a_k and measured response time $C(c^{R,i}_k)$. For write requests we define the quantity $D(c^W_{k,a_k})$.

Service demands of read requests in the queueing model are assumed to be exponentially distributed. We have found this to be a good approximation since our measurements roughly match the theoretical exponential of $CV = 1$: the CV of measured service times ranges in $CV^R_{a_k} = [0.39; 3.13]$ for all $D(c^R_{k,a_k})$. During simulations of consolidation scenarios each read request arrival $c^{R,i,n}_k$ observes the arrival queue length of requests of its own class, A^R_k, and samples an exponentially distributed service time with mean $D(c^R_{k,a_k})$, where $a_k = 1 + A^R_k$.

Population. The model is parameterized with a population of *BatchCustomers*. Before entering the queueing station each *BatchCustomer* is forked into a number of individual requests and thus represents a batch arrival. Determining the population size from measurements, e.g. the number of benchmark application threads, is difficult, since a batch of requests submitted by in-VM schedulers might consist of requests originating from multiple threads. In order to approximate the appropriate population size in consolidation based on isolation measurements only, we leverage results obtained from the previously introduced linear predictor formulas. Specifically, we predict consolidation throughputs as defined in Section 6.2.2 and then modulate the population size in simulations until simulation throughputs closely match the analytical prediction.

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

6.3.2 Model Calibration Methodology

During benchmark experiments we found that in-VM throughput measurements can significantly differ from throughputs measured at the storage server. In the absence of detailed knowledge of VMM internals we explain this behavior with optimization operations along the storage I/O path. Table 10 reveals that file server throughputs are especially affected by these operations, where measured read request throughputs at the storage server are reduced to roughly 60% compared to quantities recorded inside VMs. Throughputs for web and mail server workloads are monitored at similar quantities across the system stack.

We model this behavior with a heuristic calibration technique similar to Section 4.3.1. However, instead of parameterizing the queueing station with a static merge value ω we estimate this parameter dynamically for

Table 9 Summary of Input and Output Parameters for the Decomposition Simulation Model When Predicting Read Requests. The Notation for Write Requests is Similar.

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

Table 10 Measured Throughputs (cmd/s) in Isolation at VM and Storage Server Level.

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

each batch arrival. The ω value is based on the throughput ratios measured in isolation, as well as the size b^R of an arriving batch:

$$\omega = b^R \left(1 - \frac{T^R_{k,san}}{T^R_{k,vm}}\right), \quad T^R_{k,vm} \geq T^R_{k,san}, \tag{24}$$

where $T^R_{k,vm}$ and $T^R_{k,san}$ are the throughputs measured at VM and storage server level, respectively. Thus we merge an equivalent percentage of requests from each batch arrival as we observe throughput reductions in our isolation measurements. Table 9 offers a summary of input and output parameters for the decomposition simulation model.

7 Validation Experiments: Decomposition Model

We conduct our heterogeneous workload consolidation study using emulated disk workloads of file, web, and mail server type applications. Workloads are generated with FileBench [11], a framework for measuring and comparing file system performance. We maintain the default workload specification and use the recommended parameters for small configurations (50000 files in the initial file set). Thus read/write mixes are defined as $[0.59; 0.41]$, $[0.92; 0.08]$, and $[0.39; 0.61]$ for file, web, and mail workloads, respectively.

The presented measurements are gathered during a series of benchmarking experiments, each consisting of 15 runs of 300s length. We report results as the means over all 15 iterations or based on a representative run. Experiments are conducted with a single VM running in isolation and with two or three VMs on the same server in consolidation. For example, we consolidate one web and one mail server, denoted Web+Mail, and one web and two file servers, denoted Web+File+File. Validation results are reported as absolute relative errors of predictions compared to measurement.

7.1 Model Validation

7.1.1 Measurement Results

The performance of heterogeneous workloads is especially difficult to predict, as the joint resource usage of such workload mixes may result in volatile interference effects on disk requests. Table 12 reflects this volatility and shows highly variable measurements of throughputs in consolidation. In Web+Mail, e.g., the read throughput of the mail server workload is monitored at 132cmd/s whereas we only record 95cmd/s throughputs for mail server read requests in File+Mail. Since the difference in read throughput is only minor for the web server in both Web+Mail and Web+File consolidation cases, consolidating web and mail server workloads appears to be preferable for achieving large mail server read throughputs.

Response times are equally hard to foresee in consolidation: Mail server reads are measured at 45.5ms in File+Mail and at 39.0ms in the higher utilized scenario of Mail+Web+Web. We find response times of write requests to be significantly smaller than of read requests. This could be a result of disk request caching at the storage server. Consolidation causes latencies of both read/write request types to increase. The web server workload displays the highest rate of latency performance degradation: Read requests in Web+File+File are roughly 435% larger compared to the web server isolation measurements presented in Table 11.

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

Table 11 Mean in ms, Standard Deviation (std), and Coefficient of Variation (cv) Statistics for Read Request Service Times, Batch Sizes, Batch Think Times, Response Times, and Arrival Queue Lengths (Aqueue) as Measured in Isolation with Blktrace.

Workload	*Service Time*			*Batch Size*			*Batch Think Time*			*Response Time*			*Aqueue*		
	mean	std	cv	mean	std	cv	mean	std	cv	mean	std	cv	mean	std	cv
File	2.94	4.39	1.50	5.07	4.85	0.96	151	188	1.24	19.9	35.4	1.78	9.57	5.44	0.57
Mail	2.66	2.82	1.06	5.49	5.32	0.97	149	113	0.76	17.3	22.6	1.31	8.12	5.48	0.68
Web	1.58	2.43	1.56	5.79	6.97	1.20	141	88.4	0.63	11.8	22.7	1.92	8.39	6.91	0.82

Table 12 Measurement of Throughputs and Response Times for Consolidated Workloads.

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

Table 13 Accuracy of Read/Write Mix Predictions for Consolidated Workloads from Linear Predictors.

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

7.1.2 Prediction Accuracy of Read/Write Mixes

Table 13 reports approximation errors of read/write mix predictions for two VM and three VM consolidation scenarios obtained from the linear estimators. Results are excellent for two VM consolidations with errors ranging in $[0.02, 0.12]$. When moving to higher utilizations where three VMs are consolidated on the server modeling results slightly worsen, but still remain in a low range of $[0.01, 0.20]$. Given the blackbox view of the system, as well as the simplicity of the modeling approach the quality of read/write mix predictions is very good.

7.1.3 Prediction Accuracy of Throughputs

We show validation results for predictions of the total read/write throughput according to the linear estimator (16), as well as per-VM throughput estimates. Table 14 illustrates that approximation errors for total throughputs are equally low as when predicting read/-write mixes. Here results for two VM and three VM cases are similar and range in $[0.02, 0.13]$ and $[0.02, 0.20]$ for throughputs of read and write requests, respectively.

We are especially interested in the accuracy of per-VM throughput predictions. The decomposed simulation model leverages these estimates to approximate request class populations and so large errors for per-VM throughputs may also reflect on response time predictions of the simulation model. Our approach delivers mostly fair results < 0.30 for reads, with a single outlier of 0.39 for File+Mail. Interestingly, our model appears to capture three VM cases better than scenarios with two VMs. Approximation errors for writes range in $[0.06, 0.41]$. Notice that read requests have errors generally lower than write requests; this is a positive property since predicting the performance of read requests, which are mostly synchronous, is much more relevant than predicting the performance of write requests, which are mostly asynchronous and thus do not impact much on the response times of the applications in the VMs.

7.1.4 Prediction Accuracy of Response Times

Linear Estimators. The results in Table 15 indicate that our heuristic for response time predictions works well for read requests, which are the most important to predict; most of absolute relative errors are in a narrow interval ranging in $[0.01, 0.25]$, while a few cases show larger errors ranging in $[0.31; 0.35]$. In particular, consolidation scenarios of web server workloads appear to be difficult to predict for the linear estimators. However, in light of the above addressed 435% measured latency increase for consolidated web server workloads, such prediction errors may be satisfactory. Results for

Table 14 Accuracy of Throughput Predictions for Consolidated Workloads from Linear Predictors.

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

write request response times contain some larger error values, but apart from the difficult case of Web+File, range in $[0.002, 0.38]$. However, mean relative error values over all results compare at 0.10 and 0.18 for reads and writes, respectively.

Simulation Model. Validation of simulation results focuses on read requests. Table 15 illustrates that approximation errors of mean response times are in line with approximation errors of the analytical solution.

We choose an example case for each workload type to illustrate the potential of our model in fitting simulation results to measured distributions. Figure 13 (a) shows that predicted response times in File+Mail are pessimistic across most of the distribution for mail server requests. The quality of results is equally good for the web server workload in Figure 13 (b). We record 90th percentiles of roughly 0.0755s and 0.0726s for measurement and simulation, respectively. In the higher utilized and more difficult to predict three VM consolidation scenario illustrated in Figure 13 (c), the response time distribution of file server requests is also approximated well.

Response time predictions of write requests are more difficult due to high variability in service times with $CV^W_{a_k} = [1.03; 13.5]$ and some cases where read requests interfere with writes as depicted in Figure 10 (b). Furthermore, the often asynchronous nature of write requests [44] may be more accurately captured with an open queueing model. Our simulation model assumes that service demands of write requests follow a Hyperexponential distribution and delivers mean relative errors of 0.33 and 0.44 when predicting write request response times for two VM and three VM consolidations, respectively.

Summary. The validation confirms that the proposed approximations could be effective in several cases of practical interest. Predictions of read/write mixes and read request throughputs are of high quality. Approximating read request response times is more difficult, but again our models deliver fair results. More work is needed towards investigating the behavior of write requests, which appear to be the harder to predict in consolidation.

8 Related Work

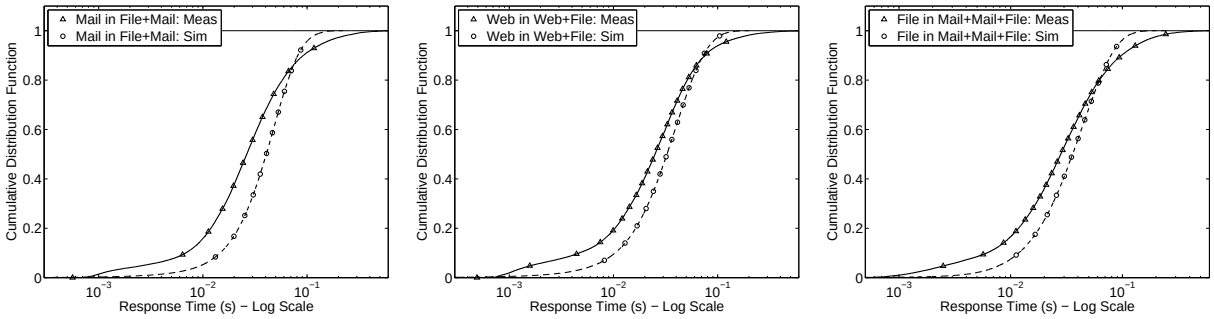
A large amount of research literature is concerned with scheduling algorithms for disk I/O in virtualized environments. The main challenges regarding scheduling are to provide fair access to the shared storage resource for all in-VM applications, while maintaining performance isolation, i.e. disk accesses by one application should not affect the I/O performance of another. This work can be structured into approaches concerned with scheduling disk access on a single VMM [32] and methods that coordinate I/O scheduling across multiple independent virtualized servers sharing a storage device [30].

Performance isolation in the presence of resource contention is studied in [34]. 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 [42]. Closer to our work, [13] derive a mathematical model to predict disk I/O throughputs when moving from a native system to an isolated VMware ESX server environment. [31] 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 [18]. One approach is to use queueing theory in order to predict performance attributes of applications when migrated from a native to a virtualized environment [17]. A shared server environment is modeled as a time-domain queueing model with GPS scheduling in [24] in order to compute and assign resource shares. In [33], layered queueing networks are used to model multi-

Table 15 Accuracy of Response Time Predictions for Consolidated Workloads from Linear Predictors and Simulation.

	Predicted with Linear Estimators						***Predicted in Simulation***		
Workload	*VM1*		*VM2*		*VM3*		*VM1*	*VM2*	*VM3*
	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_W	Δ_R	Δ_R	Δ_R
File+Mail	0.01	0.10	0.23	0.07	N/A	N/A	0.17	0.10	N/A
Web+File	0.25	0.16	0.04	0.50	N/A	N/A	0.09	0.25	N/A
Web+Mail	0.19	0.20	0.02	0.30	N/A	N/A	0.11	0.49	N/A
Mail+Web+Mail	0.15	0.10	0.30	0.03	0.12	0.14	0.06	0.21	0.04
Mail+Web+Web	0.17	0.07	0.33	0.27	0.35	0.11	0.14	0.26	0.19
Web+Web+File	0.32	0.09	0.31	0.05	0.12	0.002	0.29	0.37	0.21
Web+File+File	0.24	0.21	0.02	0.24	0.04	0.21	0.33	0.25	0.26
Mail+Mail+File	0.08	0.36	0.08	0.38	0.11	0.27	0.12	0.16	0.06



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

Fig. 13 Distributions of Measured and Simulated Response Times in Consolidation.

tier applications hosted in consolidated server environments. Recently, [37] propose an iterative model training technique based on artificial neural networks for dynamic resource allocation in consolidated virtualized environments. In [40], autonomic computing techniques and a global utility function are used to dynamically allocate CPU resources in virtualized environments. While some of the work above captures coarse grained disk requirements in the model in order to predict effects of resource allocation changes on performance of consolidated applications, none specifically tries to predict fine grained disk I/O request performance degradation due to workload consolidation.

Prediction of disk request response time granularity based on a machine learning technique is presented in [45]. 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.

9 Conclusions and Future Work

We have presented simple models that can predict response times, throughputs, and read/write mixes of disk I/O requests when consolidating workloads on to a shared storage device. Our contributions are threefold. Firstly, we parameterize the model with in-VM measurement data only instead of instrumenting the VMM. Secondly, we introduce modeling techniques for homogeneous workload consolidations that calibrate a trace-driven simulation model without detailed knowledge of VMM internal operations. Thirdly, we define simple linear estimators and a simulation model that decompose the workload into read and write requests for prediction of heterogeneous workload consolidations.

Proposed methodologies are validated against system measurements. The trace-driven simulation model produces better results than an established product form solution for response time predictions of certain homogeneous workload types. Validation experiments with emulated application workloads show that the decomposition model can accurately predict read/write mixes and throughputs for a variety of heterogeneous consolidation scenarios. Response times are more difficult to predict, but the model delivers fair results for predictions of response time means and distributions.

Our approach is limited to scenarios where fine-grained monitoring data is available for a VM workload configuration. In practice such detailed monitoring may not always be available or might be considered invasive. While our simple models work well in the presented environment, specialized layered queueing models may be beneficial when modeling more complex cloud topologies [39,33].

For future work we plan to validate the proposed methodologies on other virtualization platforms with similar system characteristics and explore application to different storage technologies. It would also be interesting to investigate how our models perform in environments where storage is federated across multiple VMMs, i.e. physical hosts. To capture more realistic disc I/O workloads we plan to move from synthetic and emulated application workloads to production traces. 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.

Acknowledgement

The work of S. Kraft has been partially funded by the InvestNI/SAP VIRTEX project. The work of G. Casale has been supported by the Imperial College Junior Research Fellowship scheme. Thanks to Stephen Dawson for valuable comments. S. Kraft is also affiliated with Queen's University Belfast.

References

1. Blktrace-Linux man page. `http://linux.die.net/man/8/blktrace`.
2. FFSB-v6.0-rc2. `http://sourceforge.net/projects/ffsb`.
3. iSCSI SAN configuration guide. `http://www.vmware.com/pdf/vi3_35/esx_3/r35u2/vi3_35_25_u2_iscsi_san_cfg.pdf`. Revision: 20090313.
4. Openfiler. `http://www.openfiler.com`.
5. Postmark-1.51-7. `http://packages.debian.org/sid/postmark`.
6. RHEL 5 IO tuning guide. `http://www.redhat.com/docs/wp/performancetuning/iotuning/index.html`.
7. Storage queues and performance. `http://communities.vmware.com/docs/DOC-6490`.
8. Tshark manpage. `http://www.wireshark.org/docs/man-pages/tshark.html`.
9. VMware. `http://www.vmware.com`.
10. Timekeeping in VMware virtual machines. Technical Report WP-065-PRD-02-01 Rev:20081017, VMware, 2008.
11. Filebench. `http://www.solarisinternals.com/wiki/index.php/FileBench`, 2010.
12. I. Ahmad. Easy and efficient disk I/O workload characterization in VMware ESX server. In *IISWC*, pages 149–158. IEEE, 2007.
13. 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.
14. J. Axboe. Linux block IO - present and future. In *Linux Symposium*, pages 51–61, 2004.
15. 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.
16. T. Begin, A. Brandwajn, B. Baynat, B. E. Wolfinger, and S. Fdida. High-level approach to modeling of observed system behavior. *Performance Evaluation*, 67(5):386 – 405, 2010.
17. 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.
18. M. N. Bennani and D. A. Menascé. Resource allocation for autonomic data centers using analytic performance models. In *ICAC*, pages 229–240. IEEE, 2005.
19. J. C. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *INFOCOM*, volume 1, pages 120–128, 1996.
20. D. Boutcher and A. Chandra. Does virtualization make disk scheduling passé? *SIGOPS OSR*, 44(1):20–24, 2010.
21. G. Casale, S. Kraft, and D. Krishnamurthy. A model of storage I/O performance interference in virtualized systems. In *DCPerf*, pages 34–39, June 2011.
22. G. Casale, N. Mi, and E. Smirni. Bound analysis of closed queueing networks with workload burstiness. In *SIGMETRICS*, pages 13–24. ACM, 2008.
23. 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.
24. A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *IWQoS*, pages 381–398. Springer, 2003.
25. A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM*, pages 1–12. ACM, 1989.
26. T. Field. JINQS: An extensible library for simulating multiclass queueing networks, v1.0 user guide. `http://www.doc.ic.ac.uk/~ajf/Research/manual.pdf`.
27. G. R. Ganger and Y. N. Patt. The process-flow model: Examining I/O performance from the system's point of view. In *SIGMETRICS*, pages 86–97. ACM, 1993.
28. S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOM*, pages 636–646, 1994.
29. 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.
30. A. Gulati, I. Ahmad, and C. A. Waldspurger. PARDA: Proportional allocation of resources for distributed storage access. In *FAST*, pages 85–98. USENIX, 2009.
31. A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In *VPACT*, 2009.
32. W. Jin, J. S. Chase, and J. Kaur. Interposed proportional sharing for a storage service utility. *ACM PEVA*, 32(1):37–48, 2004.
33. 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.
34. 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.
35. S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick. IO performance prediction in consolidated virtualized environments. In *ICPE*, pages 295–306, March 2011.

36. S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson. Estimating service resource consumption from response time measurements. In *VALUETOOLS*, pages 1–10. ICST, 2009.
37. S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application performance modeling in a virtualized environment. In *HPCA*, pages 1–10, 2010.
38. 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.
39. J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance model driven QoS guarantees and optimization in clouds. In *CLOUD '09*, pages 15–22. IEEE, 2009.
40. D. A. Menascé and M. N. Bennani. Autonomic virtualized environments. In *ICAS*, page 28, july 2006.
41. 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.
42. 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.
43. A. Riska and E. Riedel. Disk drive level workload characterization. In *USENIX*, pages 97–102, 2006.
44. C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *USENIX Winter*, pages 405–420, 1993.
45. 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.

A. Justification of Read/Write Mix Estimator

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

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

where we can identify the terms $\alpha_{i,j}^R = n_{i,j}^R/(n_{j,i}^R + n_{j,i}^W)$ and $\alpha_{j,i}^R = n_{j,i}^R/(n_{j,i}^R + n_{j,i}^W)$. Consider now the approximation $\alpha_i^R \approx \alpha_{i,j}^R$, which assumes that the relative throughput fraction of reads incoming from VM i is the same in isolation and consolidation. This approximation is accurate if the arrival process of VM i is loosely dependent on the overheads of consolidation. Using this approximation we get

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

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

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

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