Workload Decomposition for QoS in Hosted Storage Services

Lanyue Lu
Rice University, Houston, USA
ll2@rice.edu

Kshitij Doshi
Intel, USA
kshitij.a.doshi@intel.com

Peter Varman
Rice University, Houston, USA
pjv@rice.edu

ABSTRACT

The growing popularity of hosted storage services and shared storage infrastructure in data centers is driving the recent interest in performance isolation and QoS in storage systems. Due to the bursty nature of storage workloads, meeting the traditional response-time Service Level Agreements requires significant over provisioning of the server capacity. We present a graduated, distribution-based QoS specification for storage servers that provides cost benefits over traditional QoS models. Our method RTT partitions the workload to minimize the capacity required to meet response time requirements of any specified fraction of the requests.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: [Modeling techniques]; D.4.2 [**Operating Systems**]: Storage Management—*Secondary storage*

General Terms

Algorithms, Management, Performance

Keywords

Workload Decomposition, QoS, Storage Service, Capacity Provisioning, Response Time

1. INTRODUCTION

The increasing complexity of managing stored data and the economic benefits of consolidation are driving storage systems towards a service-oriented paradigm. Storage centers like Amazon S3 [1], and services by Microsoft [3] and Apple [2], already provide simple storage services for personal and corporate clients, who purchase storage space and access bandwidth to store and retrieve their data. In service-oriented computing environments, higher-level applications can leverage storage services for transparent global access to shared data, with guarantees on reliability and performance.

MW4SOC '08, December 1, 2008, Leuven, Belgium

The evolution of storage systems towards a service orientation raises a large number of issues including the choice of access protocols and granularity of access, structure of Service-Level Agreements (SLAs), performance, QoS compliance, pricing, data security, privacy, and integrity (*e.g.* see [7, 10, 13]). We assume a general framework between the client and storage service provider that is used to negotiate the pricing and service guarantees based on different QoS models. Such a model usually includes the storage space, access performance in terms of I/O bandwidth (req/sec, bytes/sec) or latency (average or maximum request response time), and reliability and availability levels for different customers.

This paper deals with the issue of providing QoS performance guarantees to storage service clients. The performance SLAs typically provide clients with minimum throughput [8, 11] guarantees, or response time bounds [9, 12] for rate-controlled clients. The server must provision sufficient resources to ensure that the clients receive their stipulated performance, and ensure that at run-time the clients are insulated from each other so that runaway or malicious behavior by some client does not affect the guarantees made to the rest of them. A difficult problem that has hindered the adoption of QoS for storage systems (reflected in the rudimentary nature of storage services) is the pessimistic capacity estimates required to guarantee typical SLAs. Unlike networks, dropping requests is not a viable option for storage systems whose protocols do not support re-try mechanisms; throttling mechanisms often result in underutilized resources and are also unsuitable for open workloads. Since many storage workloads tend to be bursty (*i.e.*, the instantaneous arrival rates in some periods is significantly higher than the long-term rate), capacity estimates based on worst-case patterns result in gross over provisioning of the server. Service providers are understandably reluctant to commit to guaranteed QoS if it requires provisioning resources greatly in excess of the typical or average service requirements.

In this paper we focus on improving capacity provisioning by shaping storage workloads to provide graduated, distribution based QoS guarantees. In a graduated approach, rather than specifying a single response time bound for all the requests, the performance SLA is specified by a distribution of response times. For instance, rather than specifying a single upper bound r on the response time for all requests, a client may relax the requirements and instead require that 95% of the requests meet the bound r and the remaining requests meet a more relaxed latency r'. This approach can provide significant benefits to the service provider since the worst-case requirements are usually determined by the tail of the

workload. Our results confirm the existence of a well-defined knee in the QoS-capacity relation, whereby an overwhelming portion of the server capacity is used to guarantee the performance of a small fraction of the requests. By relaxing the performance guarantees for this small fraction, a significant reduction in server capacity can be achieved while maintaining strong QoS guarantees for most of the workload. The server can pass on these savings by providing a variety of SLAs and pricing options to the client. Most clients are willing to accept such a graduated service, provided the degradation is quantifiable and bounded, and is priced appropriately. Storage service subscribers that have highly streamlined request behavior, and who therefore require negligible surplus capacity in order to meet their deadlines, can be offered service on concesssional terms as reward for their "well-behaved-ness."

The rest of the paper is organized as follows. Section 2 presents the model and workload decomposition algorithms. We show that an intuitive algorithm based on pruning requests from intervals of maximum load is not as effective in reducing capacity requirements as our dynamic decomposition scheme RTT. Simulation results of response time distributions for several block-level storage workloads are presented in Section 3. The paper concludes with Section 4.

2. OVERVIEW AND SYSTEM MODEL

In our model, SLAs are specified by a Response Time Distribution (RTD) indicating the minimum fraction f_i of the workload that must meet a given response time bound R_i. An n-tier RTD is represented by the set of n pairs of frequency and response times: $\{ (f_i, R_i) \mid 1 \leq i \leq n, 0 < f_{i-1} < f_i \leq 1, f_n = 1.0\}$. Hence the RTD specifies a lower bound on the cumulative response time distribution achieved by that workload. A traditional SLA in which 100% of the requests are guaranteed a maximum latency of 200ms corresponds to the RTD $(1.0, 200\text{ms})$. A 3-tier RTD $\{(0.9, 20\text{ms}), (0.99, 50\text{ms}), (1.0, 500\text{ms})\}$ indicates that no more than 10% of the requests can exceed 20ms latency, no more than 1% should exceed 50ms, while all requests must be served within a 500ms response time.

Figure 1 shows the organization of an n-tier RTD. An application's request stream arriving at the workload shaper is partitioned into service classes W_1 through W_n, and directed to separate queues. The queues are multiplexed on the server using an isolating QoS scheduler like pClock [9], that provides non-interfering response time guarantees to each class. In this paper we will concentrate on a two-tier RTD architecture for simplicity; the extension to multiple QoS tiers is deferred to our future work. In this case, the two service classes W_1 and W_2 will be referred to as *primary* and *secondary* classes respectively.

The workload shaper classifies the requests into primary and secondary classes using a decomposition algorithm. In earlier works, different classification criteria were used in profiling of CPU [14] and network [6] workloads. In the context of CPU resource overbooking [14], intervals with highest CPU utilization were classified as secondary. In [6], end-to-end measured response times of a networked server were used for partitioning, and the highest response times requests are classified as secondary. Both these algorithms are static methods that are used for off-line analysis of the performance trace. We evaluate two algorithms, Min-Max Load (MML) and Response Time Threshold (RTT), to partition the workload.

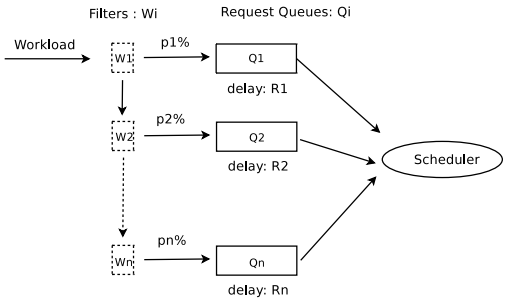

Figure 1: Architecture of a multi-queue scheduler providing graduated QoS guarantees. The request stream is partitioned into classes W_1 to W_n with different response times.

Min-Max Load (MML): MML is a natural and intuitively appealing method based on the utilization approach of [14]. Time is divided into fixed-size windows, and the load (number of requests) in each window is monitored. The approach minimizes the maximum load in any interval, by iteratively moving requests out from the currently highest loaded window until the desired fraction f of the requests remains; the requests that are removed form the secondary class. The remaining requests form the primary class.

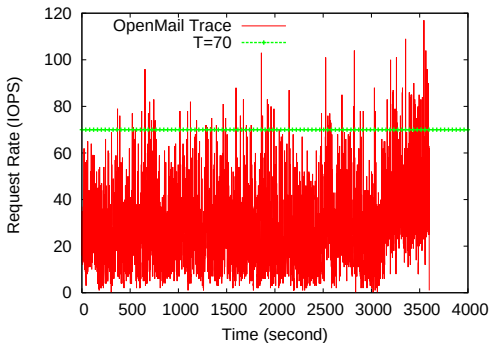

Figure 2: MML decomposition for OpenMail trace

Figure 2 shows the a portion of an OpenMail trace of I/O requests using a time window of 1s. Note that the peak request rate is about 120 IOPS while the average request rate is only about 26 IOPS, and the request rate in some intervals is zero. If the workload is pruned at 70 IOPs as shown, all requests below (above) the horizontal line will belong to the primary (secondary) class.

Response Time Threshold (RTT): This method explicitly incorporates the response time requirements into the workload shaping policy, so it can accurately identify the requests that will miss their deadlines. In Figure 3, the Cumulative Arrival Curve (AC) shows the total number of requests that have arrived at any time. An arriving request causes a jump in AC at the arrival instant. The Response Time Curve (RC) is obtained by time-shifting the AC by the latency bound d; it indicates the time by which a request must complete to meet its latency guarantee. The Service Curve (SC) models a constant rate server with a slope equal to its capacity as long as the queue is not empty. Its value at any

time is the total service provided till that time. Whenever SC crosses the RT (*i.e.* RT falls to the left of SC), it indicates that a request will miss its deadline; hence either this request or one of the previous requests must be moved to the secondary class in order to make RT fall below SC.

For a given server capacity S and response time requirement R, RTT will move the smallest number of requests possible to the secondary class. It does so by monitoring the length of the queue at the server; if the number of outstanding requests is higher than the threshold $L = S \times R$, the incoming request will be redirected to the secondary queue. This is because this request (assuming FCFS scheduling of the queue) will experience a delay more than the latency bound R, and will not meet its response time guarantee. With a different queueing service discipline, some other queued request will miss its deadline, since queue overflow means there is insufficient capacity for all these requests to meet their deadlines. The request that overflowed the primary queue is moved to the secondary class with a more relaxed response time requirement.

To determine the minimum capacity required for a specified fraction f of the workload to meet the deadlines the workload is profiled using the strategy above for different values of the capacity. For a given choice of server capacity, the percentage of requests that need to be moved to the secondary class is found. By systematically exploring the space of capacities using a binary-search strategy, the minimum capacity required to meet the given fraction f is found efficiently.

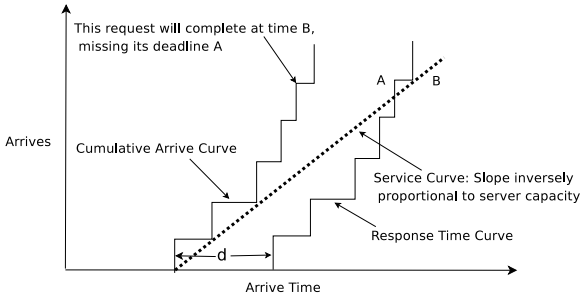

Figure 3: RTT decomposition of Workload

3. PERFORMANCE EVALUATION

3.1 Overview of Experiments

We measured the characteristics of three different storage applications, Financial Transactions (FT), OpenMail (OM), and WebSearch (WS), using traces obtained from UMass Storage Repository [5] and HP Research Labs [4]. All of these are low-level block storage I/O traces. The FT traces are from an OLTP application running at two large financial institutions, which processes financial transactions. OpenMail traces are collected from HP email servers during the servers' busy periods. The WebSearch traces are from a popular search engine, which consists of the web search requests from users.

Altogether there were 2 FT traces (FT1, FT2), 6 OM traces (OM1 to OM6), and three WS traces (WS1 to WS3). While our experiments were performed for all traces, only some of the results are discussed in detail here for reasons of space. We conducted three sets of experiments to evaluate the QoS-capacity tradeoffs in the workloads as detailed in the following sections. The first set in Section 3.2 compares MML and RTT as workload decomposition algorithms. In Section 3.3 the the relation between server capacity and percentage of requests guaranteed primary class service is presented. The third set of experiments described in Section 3.4 analyzes the tail of the distribution and the tradeoffs in handling these secondary class requests.

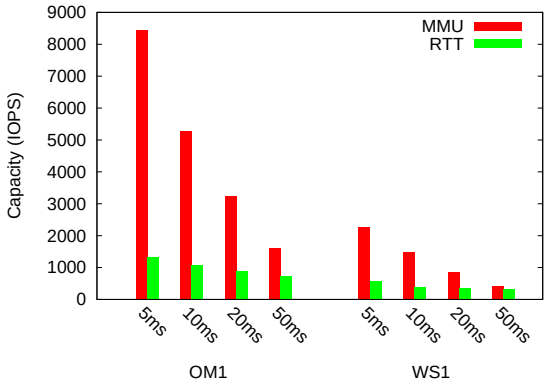

Figure 4: Capacity comparison for MML and RTT using OpenMail(OM) and WebSearch(WS) traces, for 90% of the workload to meet the latency bound.

3.2 Comparison of MML and RTT

In the first set of experiments, we compare the MML and RTT decomposition methods for workload partitioning. We used two traces, specifically OM1 and WS1 from the Open Mail and Web Search applications respectively. We plot the minimum server capacity required to guarantee that 90% of the workload meets a specified response time. Figure 4 shows the capacity for the two applications for response times of 5, 10, 20, and 50ms. MML uses a time window of 100 ms, and randomly removes requests in each overloaded interval to reach the desired threshold. In each case it can be seen that RTT requires significantly less capacity than MML to guarantee the same response time bound. Similar results were obtained over the range of primary class sizes and traces in the data set. The experiments demonstrate that in the environment of asynchronous request arrivals, partitioning the workload using the dynamic response-time aware RTT method is clearly advantageous over using static load measures. By experimenting with a wide range of time-window sizes (from tenths of a ms to 100 ms), we found MML is very sensitive to the window size; the capacity requirement first decreases and increases again as the size of time window decreases. Choosing the optimal window size is hard in practice; even with the optimal window size, the resulting capacity is still much higher than RTT. The main reason is that, unlike RTT, MML does not consider the queueing effects when it chooses requests to downgrade during a burst. Hence, it may unnecessarily move requests that do not actually cause a deadline miss, while retaining those that will require extra capacity to meet their deadlines.

3.3 Capacity vs Response Time Guarantees

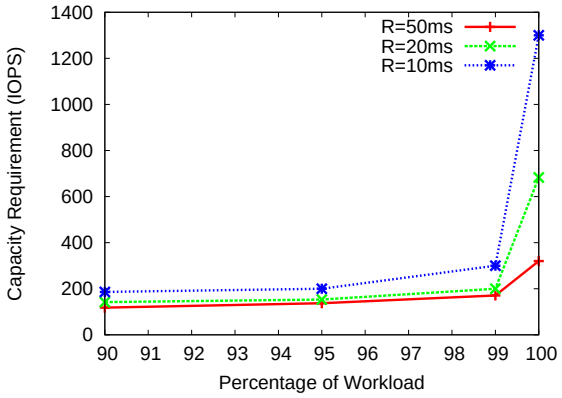

Figure 5: FT2: Capacity vs Percentage of Workload between 90% to 100% meeting specified latency bound

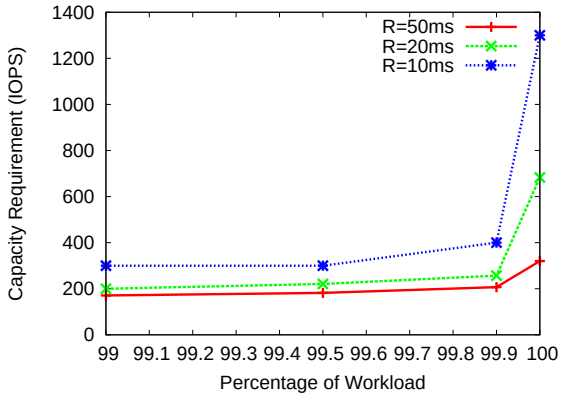

Figure 6: FT2: Capacity vs Percentage of Workload between 99% to 100% meeting specified latency bound

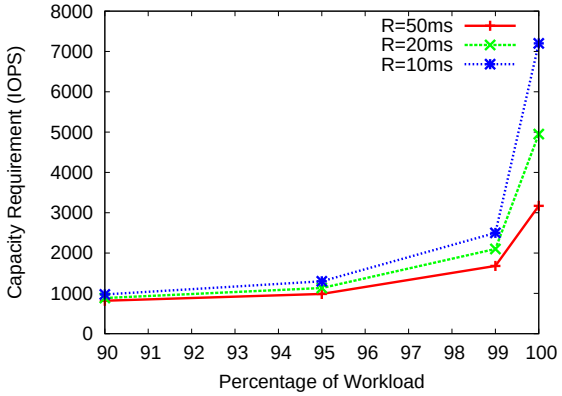

Figure 7: OM5: Capacity vs Percentage of Workload between 90% to 100% meeting specified latency bound

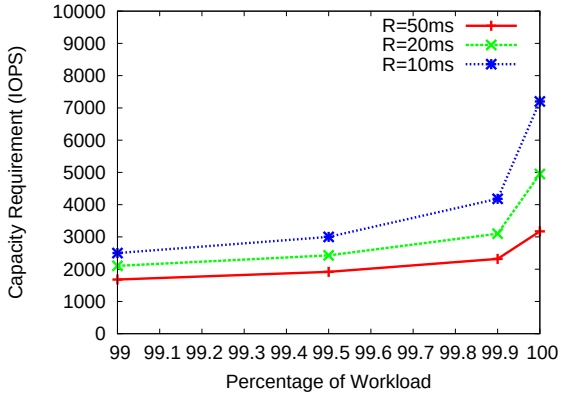

Figure 8: OM5: Capacity vs Percentage of Workload between 99% to 100% meeting specified latency bound

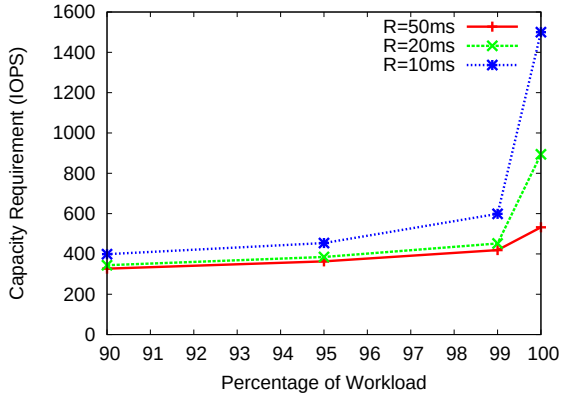

Figure 9: WS1: Capacity vs Percentage of Workload between 90% to 100% meeting specified latency bound

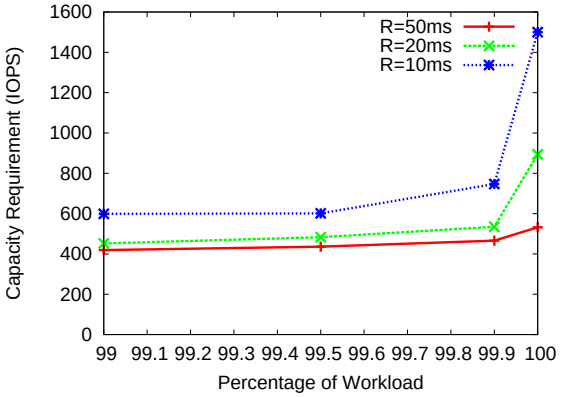

Figure 10: WS1: Capacity vs Percentage of Workload between 99% to 100% meeting specified latency bound

This set of experiments explores the tradeoffs between the fraction f of the workload in the primary class, its latency bound R, and the minimum required capacity S. When $f = 100\%$, this gives the minimum capacity required for all the requests to meet the latency bound R. As f is relaxed, a smaller capacity is sufficient for a fraction f of the workload to meet the response time bounds.

Figures 5 and 6 show the QoS variation of the FT2 workload as the capacity is varied. It plots the capacity needed for a fraction f of the workload to meet response times bounds of 50ms, 20ms and 10ms. Figure 5 covers the range of f between 90% and 100%, while Figure 6 shows a finer-grained distribution with f ranging from 99% to 100%. As can be seen in Figure 5, the capacity required falls off significantly by exempting between 1% and 10% of the workload from the response time guarantee. For a 10ms response time, the capacity required increases almost 7 times (from 186 IOPS to 1300 IOPS) when the percentage that is guaranteed 10ms latency increases from 90% to 100%. In going from 99% to 100% the capacity required increases by a factor of 4.3, from 300 IOPS to 1300 IOPS. For response times of 20ms and 50ms the capacity also increases by significant, though smaller, factors. In order to guarantee 20ms and 50ms latencies for the last 10% of the workload the capacity needs to be increased by factors of roughly 4.8 and 2.7 respectively, while guaranteeing the last 1% of the workload requires the capacity to increase 3.4-fold and 1.9-fold times respectively. The extent of burstiness of the workload can be gauged by looking at Figure 6, which tracks the capacity needed to guarantee the response time for between 99% and 100% of the workload. For response times of 10ms, 20ms and 50ms, increasing f from 99.9% to 100% requires capacity increases by factors of 1.9, 1.7, and 1.5 respectively. Similar trends for FT1 are noted from the data in Table 1.

Corresponding plots are shown in Figures 7 and 8 for the Open Mail workload (OM5) and Figures 9 and 10 for the Web Search trace (WS1). For the OM5 trace, for 10ms response times the increases in capacity in going to 100% from 90%, 99% and 99.9% are by factors of 7.4, 2.9 and 1.7 respectively; the corresponding numbers for the WS1 trace are 3.7, 2.5 and 2.0. Tables 2 and 3 show the results for OM1 and OM3 traces respectively, whiles the results for WS2 and WS3 are shown in Tables 4 and 5 respectively.

We note that exempting even a small fraction of the workload from the response time guarantees can substantially reduce the capacity required, specially in the case of aggressive service with low response time requirements. With more relaxed response times, both the absolute capacities and the percentage savings are smaller.

3.4 Tail of the Distribution

In this set of experiments we first measured the performance of the unpartitioned workload using a First-Come-First-Serve (FCFS) scheduler for all the requests to serve as a baseline. The cumulative response time distribution obtained for the unpartitioned workloads using FCFS scheduling is shown in Figure 11. The capacity for each workload was chosen so that 90% of the workload could meet a 10ms response time using RTT. At a capacity of 399 IOPS, only 42% of the unpartitioned WS1 workload meet a 10ms bound. In contrast, for the same trace and same capacity, when the workload was partitioned using RTT into primary and secondary classes, 90% of the workload met the 10ms latency bound (see Figure 9). The cumulative distribution rises slowly with increasing response time; in fact, only 94% of the requests meet a 1000ms latency bound. The unpartitioned workload reached 90% compliance only when the response time was around 800ms.

A similar behavior is shown by the OM5 workload for a 10ms response time bound and a capacity of 975 IOPS. In the unpartitioned workload, only 48% of the requests meet the 10ms response time bound, and reaches a 90% compliance at around 800ms. In contrast, the RTT-partitioned workload achieves 90% compliance with the 10ms latency (see Figure 7). For the FT2 workload, a capacity of 186 IOPS resulted in 59% of the unpartitioned workload, and 90% of the partitioned workload meeting the 10ms response time bound. Unlike the other two applications, the tail of the distribution is much steeper, and the unpartitioned workload had a 90% compliance at around 55ms.

The final plot shown in Figure 12 compares the tail of the response time distribution of the unpartitioned OM5 workload with that of the partitioned workload for a capacity of 1400 IOPS, split statically into a fixed 1300 IOPS for the primary group and 100 IOPS for the secondary group. As can be seen, 95% of the partitioned workload can be guaranteed a 10ms response time, in contrast to 85% for the unpartitioned case. Even with this static capacity allocation, response time is degraded only for a small percentage of the tail. In practice, a work-conserving scheduler will allocate the unused capacity of the primary class to the secondary class, and the response time distribution of the last few percent of the partitioned scheme will approach that of the FCFS curve, while maintaining its guaranteed advantage for the initial 85% of the workload. In a multi-client environment, if we provide separate capacity for different parts of the workload for each client, and schedule them in isolation using a latency aware fair-scheduler [9], then the performance of all of them can be guaranteed.

4. CONCLUSIONS

The bursty nature of storage workloads requires significant over provisioning of the capacity of storage servers to meet traditional response-time QoS guarantees. In this paper we proposed graduated, distribution-based QoS as a means to reduce server capacity provisioning, and presented a workload decomposition algorithm RTT to dynamically shape the workload to meet graduated QoS requirements. We analyzed several block-level storage workload traces to evaluate the advantages of providing graduated QoS guarantees to an application. We show that providing strong response time guarantees to only 90%-99% of the workload (instead of the entire 100%) can reduce server capacity requirements significantly, by several hundred percent in some cases. In fact, even relaxing the guarantees of just the last 0.1% of the workload results in significant reduction in the capacity required. We are continuing work on alternative strategies for improving the response times of the tail of the distribution, n-tier RTD for $n > 2$, and integrating the workload shaper with the run-time QoS scheduler.

5. ACKNOWLEDGMENTS

NSF grant CNS 0615376 is gratefully acknowledged. We thank the reviewers for their useful and detailed comments.

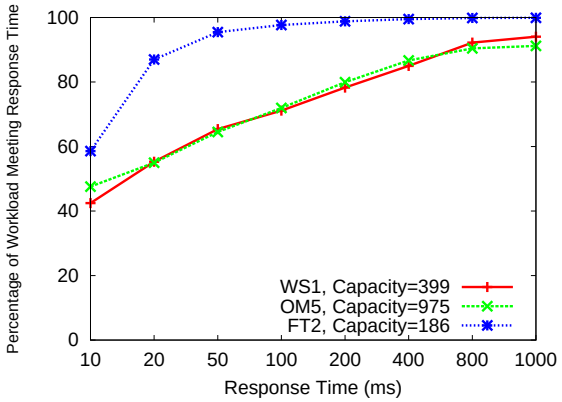

Figure 11: Tail of Response Time Distribution for FCFS Scheduling

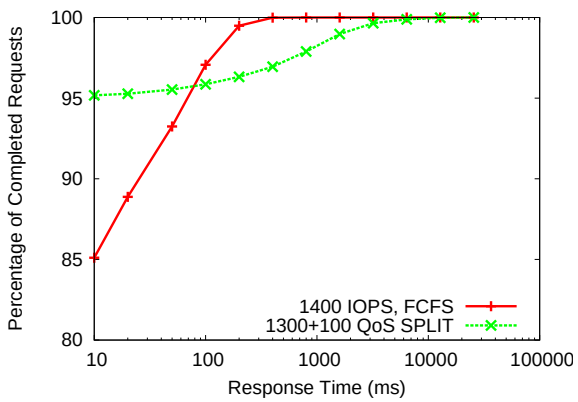

Figure 12: FCFS vs Two-tier scheduling: Response Time Distribution

Percentage	90.0	95.0	99.0	99.5	99.9	100
R = 10ms	400	599	899	1000	1200	2400
R = 20ms	256	349	500	550	683	1300
R = 50ms	179	209	273	299	358	620

Table 1: FT1: Capacity required for percentage of workload with response time R (IOPS)

Percentage	90	95	99	99.5	99.9	100
R = 10ms	1073	1594	2935	3510	4800	9161
R = 20ms	898	1314	2350	2724	3460	5750
R = 50ms	740	1040	1800	2043	2493	3640

Table 2: OM1: Capacity required for percentage of workload with response time R (IOPS)

Percentage	90.0	95.0	99.0	99.5	99.9	100
R = 10ms	1272	1900	3673	4399	5699	9468
R = 20ms	999	1450	2550	2999	3899	5500
R = 50ms	710	1039	1699	1940	2545	4098

Table 3: OM3: Capacity required for percentage of workload with response time R (IOPS)

Percentage	90.0	95.0	99.0	99.5	99.9	100
R = 10ms	375	415	543	600	700	1200
R = 20ms	320	362	441	461	511	700
R = 50ms	305	343	404	423	457	540

Table 4: WS2: Capacity required for percentage of workload with response time R (IOPS)

Percentage	90.0	95.0	99.0	99.5	99.9	100
R = 10ms	292	308	400	470	542	1000
R = 20ms	231	265	335	353	401	712
R = 50ms	213	244	300	319	358	560

Table 5: WS3: Capacity required for percentage of workload with response time R (IOPS)

6. REFERENCES

[1] Amazon simple storage service (amazon s3). http://www.amazon.com/gp/browse.html?node=16427261.

[2] Mac mobile me. http://www.apple.com/mobileme/features/idisk.html.

[3] Windows live sky dive. http://skydrive.live.com/.

[4] Public software (storage systems department at hp labs), 2007. http://tesla.hpl.hp.com/publicsoftware/.

[5] Storage performance council (umass trace repository), 2007. http://traces.cs.umass.edu/index.php/Storage.

[6] T. Dumitras and P. Narasimhan. Fault-tolerant middleware and the magical 1%. In *Proc. 6th International Middleware Conference*, 2005.

[7] T. Gao and et. al. Toward qos analysis of adaptive service-oriented architecture. In *IEEE Intl. Wksp. Service Oriented Sys. Enginr.(SOSE)*, 2005.

[8] P. Goyal, H. M. Vin, and H. Cheng. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Trans. Netw.*, 5(5):690–704, 1997.

[9] A. Gulati, A. Merchant, and P. Varman. *p*Clock: An arrival curve based approach for QoS in shared storage systems. In *Proc. ACM Intl Conf on Measurement and Modeling of Comp. Sys. (SIGMETRICS)*, 2007.

[10] R. Jurca, W. Binder, and B. Faltings. Reliable qos monitoring based on client feedback. In *Proceedings of the ACM WWW 2007 Conference*, May 2007.

[11] C. Lumb, A. Merchant, and G. Alvarez. Façade: Virtual storage devices with performance guarantees. *File and Storage Technologies (FAST)*, pages 131–144, March 2003.

[12] H. Sariowan, R. L. Cruz, and G. C. Polyzos. Scheduling for quality of service guarantees via service curves. In *Proc. of the Intl. Conf. on Comp. Comm. and Networks*, pages 512–520, 1995.

[13] A. Sheth and et al. Qos for service-oriented middleware. In *Proc. Conf. Systemics, Cybernetics and Informatics*, July 2002.

[14] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proc. of the 5th OSDI*, December 2002.