Nested QoS: Providing Flexible Performance in Shared IO Environment

Hui Wang hw5@rice.edu
Peter Varman pjv@rice.edu
Rice University, USA

Abstract

The increasing popularity of storage and server consolidation introduces new challenges for resource management. In this paper we propose a Nested QoS service model that offers multiple response time guarantees for a workload based on its burstiness. The client workload is filtered into classes based on the Service Level Objective (SLO) and scheduled to provide requests in each class a stipulated response time guarantee. The Nested QoS model provides an intuitive, enforceable, and verifiable SLO between provider and client. The server capacity in the nested model is reduced significantly over a traditional SLO while the performance is only marginally affected.

1 Introduction

Large virtualized data centers that multiplex shared resources among hundreds of paying clients form the backbone of the growing cloud IT infrastructure. The increased use of VM-based server consolidation in such data centers introduces new challenges for resource management, capacity provisioning, and guaranteeing application performance. Service Level Objectives (SLOs) are employed to assure clients a level of performance QoS like minimum throughput or maximum response time. The server should provide sufficient capacity to meet the stipulated QoS goals, while avoiding over-provisioning that leads to increased infrastructure and operational costs. Accurate provisioning is complicated by the bursty nature of storage workloads [15, 9] and sharing by multiple client VMs. Performance SLOs range from simply providing a specified floor on average throughput (e.g. IOPS) to providing guarantees on the response time of requests. The former can be readily supported using weighted Fair Queuing approaches (see Section 6); providing response-time guarantees requires that the input stream be suitably constrained.

In this paper we propose a Nested QoS service model that offers a spectrum of response time guarantees based on the burstiness of the workload. It formalizes the observation that a disproportionate fraction of server capacity is used to handle the small tail of highly bursty requests. In [14] we described a workload decomposition scheme to identify and schedule these requests to reduce capacity. However, this framework is not backed by a formal underlying SLO model; the difficulty is in coming up with a suitable specification. For instance, while the client may be satisfied with an agreement that guarantees 95% of its requests will have a response time of less than 20ms, the provider can only make such an assurance if the workload satisfies some constraints on burstiness and throughput. Further, the model should be intuitive, easy to enforce, and mutually verifiable in case of dispute. The Nested QoS model provides a formal (but intuitive and enforceable) way to specify the notion of graduated QoS, where a single client's SLO is specified in the form of a spectrum of response times rather than a single worst-case guarantee. The model properly generalizes SLOs based on a single response time guarantee (e.g. [10]).

In Section 2 we describe the Nested QoS model and its implementation. In Sections 3 and 4 we demonstrate how it can reduce capacity in single client and shared client environments respectively. It's still useful in a non-oversubscribed environment because of economy reason (improved server utilization and reduced cost). Analysis of the server capacity is presented in Section 5. Our work is related to the ideas of differentiated service classes in computer networks [4] [12] [16]. However, we believe our model and analysis are different from these works, and the decomposition and evaluation of storage traces is new.

2 System Model

The workload W of a client consists of a sequence of requests. Figure 1 shows the framework of our Nested QoS service model. The performance SLO is determined by multiple nested classes $C_1, C_2 \cdots C_n$. Class C_i is specified by three parameters: $(\sigma_i, \rho_i, \delta_i)$, where (σ_i, ρ_i) are token bucket [16] parameters and δ_i is the response time guarantee. C_i consists of the maximally-sized subsequence of requests of W that is compliant with a (σ_i, ρ_i) token bucket: that is, the number of requests in any interval of length t is

upper bounded by $\sigma_i + \rho_i t$, and no other request of W can be added to the sequence without violating the constraint. The token bucket provides an envelope on the traffic admitted to each class by limiting its burst size (σ_i) and arrival rate (ρ_i). All requests in C_i have a response time limit of δ_i. Nesting requires that $\sigma_i \leq \sigma_{i+1}, \rho_i \leq \rho_{i+1}$ and $\delta_i \leq \delta_{i+1}$.

For example, a 3-class Nested QoS model (30, 120 IOPS, 500ms), (20, 110 IOPS, 50ms), (10, 100 IOPS, 5ms) indicates that: all the requests in the workload that lie within the (10, 100 IOPS) envelope have a response time guarantee of 5ms; the requests within the less restrictive (20, 110 IOPS) arrival constraint have a latency bound of 50ms, while those conforming to the (30, 120 IOPS) arrival bound have a latency limit of 500ms.

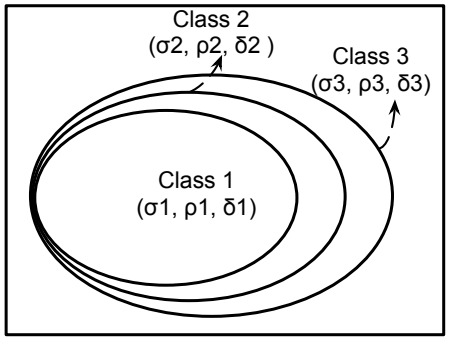


Figure 1. Nested QoS Framework

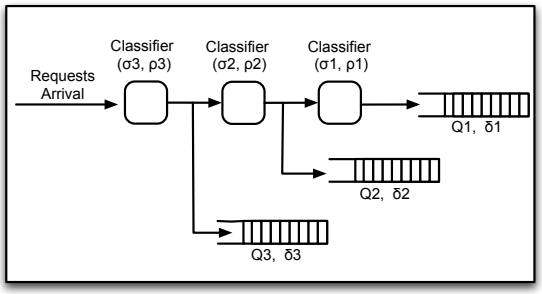


Figure 2. Nested Traffic Envelopes

Figure 2 shows an implementation of the model. It consists of two components: *request classification* and *request scheduling* (not shown in the figure). The former is implemented using a cascade of token buckets, $B_1, B_2 \cdots B_n$ (innermost is B_1). The buckets filter the arriving workload so that queue Q_1 receives all requests of class C_1, Q_2 receives requests of $C_2 - C_1$, and Q_3 receives requests of $C_3 - C_2$. By ensuring that requests in queue Q_i meet a response time of δ_i, the SLO of the Nested QoS model can be met. The scheduler services requests across the queues within a client based on their deadlines using an Earliest Deadline First (EDF) policy. To give an example of request classification, Figure 3 shows the filtering of the Exchange workload as it goes through the token bucket network.

B_i has parameters (σ_i, ρ_i) that regulates the number of requests that pass through it in any interval. Initially B_i has σ_i tokens; an arriving request removes a token from the bucket (if there is one) and passes thorough to B_{i-1} (or Q_1 if i is 1); if there are no tokens in B_i the request goes into the queue Q_{i+1} instead. B_i is filed with tokens at a constant rate ρ_i, but the maximum number of tokens is capped at σ_i. In Section 5 we compute the capacity required to meet the SLO specified by the Nested QoS model parameters.

3 Nested QoS for a Single Workload

We describe how the Nested QoS parameters of a workload will typically be determined. The client first decides the number of classes and their sizes (as a fraction of workload size) by empirically profiling the workload to achieve a satisfactory tradeoff between capacity required (cost) and performance. (Usually three classes appear to be sufficient over a variety of workloads.) Using a decomposition algorithm (see [14]) one can determine the minimum capacity κ_1 required for a fraction f_1 of the workload to meet the deadline δ_1. We choose $\rho_1 = \kappa_1$ and $\sigma_1 = \rho_1\delta_1$. We similarly profile each of the classes, and set $\rho_2 = max\{\kappa_1, \kappa_2\}$ and $\sigma_1 \leq \sigma_2 \leq \rho_2\delta_2$, and $\rho_3 = max\{\kappa_2, \kappa_3\}$ and $\sigma_2 \leq \sigma_3 \leq \rho_3\delta_3$.

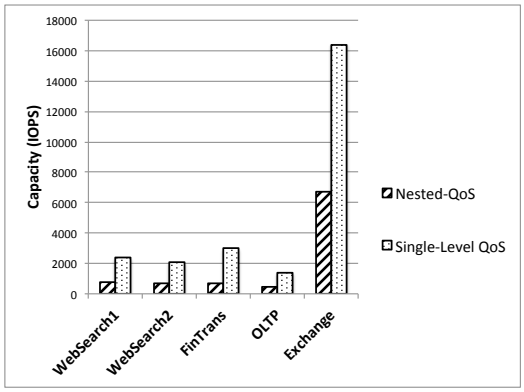
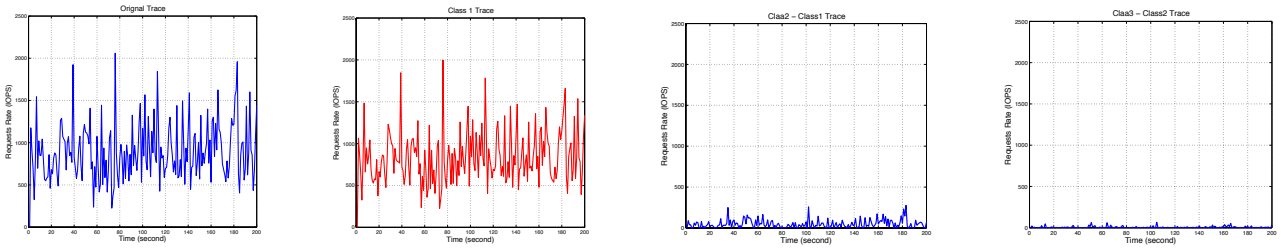


Figure 4. Capacity Requirement for Nested QoS and Single level QoS

We implemented the Nested QoS model in a process-driven system simulator and evaluated the performance separately with five block-level storage workloads (W1-W5) from UMass Storage Repository [1] and SNIA IOTTA Repository [2]: WebSearch1, WebSearch2, FinTrans, OLTP, Exchange. WebSearch1, WebSearch2 are traces from a web search engine and consist of user web



(a) Original workload (b) Workload in Queue 1 (c) Workload in Queue 2 (d) Workload in Queue 3

Figure 3. Decomposition of workload into different classes

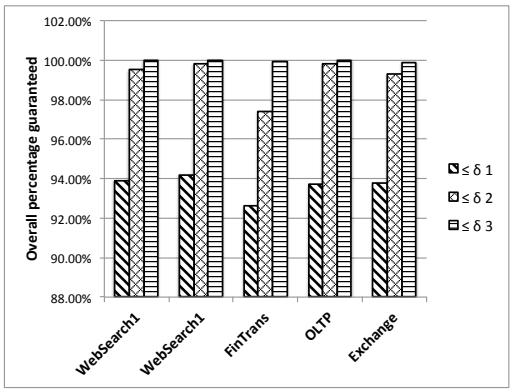


Figure 5. Performance for Nested QoS

search requests. FinTrans, OLTP are traces generated by financial transactions running at large financial institutions. Exchange trace is from Microsoft Exchange Server. The parameters for each workload are as follows: δ_1 = 5ms (all workloads); $\sigma_1 = 3$ (W1,W2,W3), $\sigma_1 = 4$ (W4), and $\sigma_1 = 33$ (W5); $\rho_1 = 650$ (W1, W2), $\rho_1 = 600$ (W3), $\rho_1 = 400$ (W4) and $\rho_1 = 6600$ (W5). For the other classes, $\sigma_{i+1} = 2\sigma_i$, $\delta_{i+1} = 10\delta_i$ and $\rho_{i+1} = \rho_i$. The values were found by profiling the workloads to guarantee more than 90% requests in C_1.

Figure 4 compares the capacity required by the workloads for the Nested and Single-Level QoS models. The capacity is significantly reduced by spreading the requests over multiple classes. Figure 5 shows the distribution of response times. In each case a large percentage (92%+) of the workload meets the 5ms response time bound, and a tiny 0.1% or less requires more than 50ms. The capacity required for Nested QoS is several times smaller than that for a Single-Level QoS, while the service seen by the clients is only minimally degraded .

4 Nested QoS for Concurrent Workloads

In a shared environment, each VM workload is independently decomposed into classes based on its Nested QoS parameters. The server provides capacity κ_j for VM j based on its capacity estimate using the formula of Section 5, and provisions a total capacity of $\Sigma_j \kappa_j$. A standard Fair Scheduler allocates the capacity to each VM in proportion to its κ_j. When VM j is scheduled it chooses a request from its queues with the smallest deadline. Figure 6 shows the organization for serving multiple clients.

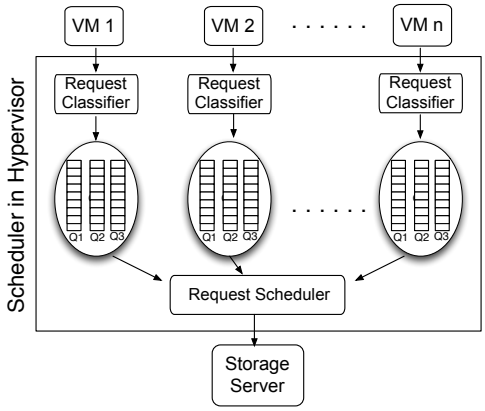


Figure 6. The architecture of Nested QoS model in a VM Environment

We compare the performance of WF2Q [3], and pClock [10] with scheduling of requested streams decomposed by Nested QoS. The former two methods try to provide 100% guarantees and their performance degrades appreciably if the capacity provisioned is less than that required. We employ two concurrent workloads WebSearch and FinTrans from two VMs. The parameters setting are: $\sigma_1 = 16$, $\rho_1 = 320$, $\delta_1 = 50ms$ for WebSearch, and $\sigma_1 = 8$, $\rho_1 = 160$, $\delta_1 = 50ms$ for FinTrans. For both workloads, $\sigma_{i+1} = 2\sigma_i$, $\delta_{i+1} = 10\delta_i$, $\rho_{i+1} = \rho_i$. Fig-

ures 7 (a) (b) show the performance of the three schedulers with the server capacity of 528 IOPS. The weight of the two VMs is 2 : 1 based on their capacity requirements. Figure 7(c) for the WebSearch workload shows that with Nested QoS 97% of the workload meets the $50ms$ response time bound, while *p*Clock and WF2Q can only guarantee 70% and 48% respectively. Figure 7(d) shows a similar performance for FinTrans. We see that Nested QoS can provide better performance guarantees than both pClock and WF2Q.

5 Analysis

The workload W consists of a sequence of requests arriving at times $1, 2, 3, ...$ The decomposition splits W into classes $C_1, C_2, \cdots C_n$. C_i consists of the requests of W that are output by the token bucket B_i. All requests in C_i have a response time no more than δ_i. From the nested definition, we require that $\sigma_i \leq \sigma_{i+1}$, $\rho_i \leq \rho_{i+1}$ and $\delta_i \leq \delta_{i+1}$. The problem is to estimate the server capacity required to meet the SLO. We present here only the result for special case where all ρ_i are equal to ρ; this is also the case used in all the experimental evaluations in this paper. We define a *busy period* to be an interval in which there are one or more requests in the system.

Lemma 1 The capacity required for all requests to meet their deadlines in the Nested QoS model, when all ρ_i are equal to ρ, is given by: $max_{1 \leq j \leq n}\{\sigma_j/\delta_j + \rho(1 - \delta_1/\delta_j), \rho\}$.

Proof: We bound the maximum number of requests that need to finish by time $t \geq 0$, where $t = 0$ is the start of the busy period. All requests with a deadline t or less must have arrived in the interval $[0, t - \delta_1]$, since δ_1 is the smallest response time bound. Requests with response time bound δ_j have, by definition, been passed by B_j. The maximum number of requests with deadline t that could have been admitted by B_j in $[0, t - \delta_1]$ is $N_j(t) = \sigma_j + \rho(t - \delta_1)$. The server capacity κ_j required to finish the $N_j(t)$ requests by t is no more than $N_j(t)/t = (\sigma_j - \rho\delta_1)/t + \rho$. First, if $\sigma_j < \rho\delta_1$ then κ_j is no more than ρ. Otherwise, we consider two cases: $t \geq \delta_j$ and $t < \delta_j$. If $t \geq \delta_j$ the maximum value of $N_j(t)/t$ is reached for $t = \delta_j$ and we get κ_j no more than $\sigma_j/\delta_j + \rho(1 - \delta_1/\delta_j)$. If $t < \delta_j$, all the requests admitted by B_j have a deadline less than δ_j and hence must belong to class C_{j-1} or smaller; in this case κ_j equals S_{j-1}. Putting the cases together, the Lemma follows. The following workload shows the capacity estimate is tight: a burst of σ_n requests at $t = 0$ followed by requests arriving at the uniform rate ρ, will require the capacity estimated by the Lemma.

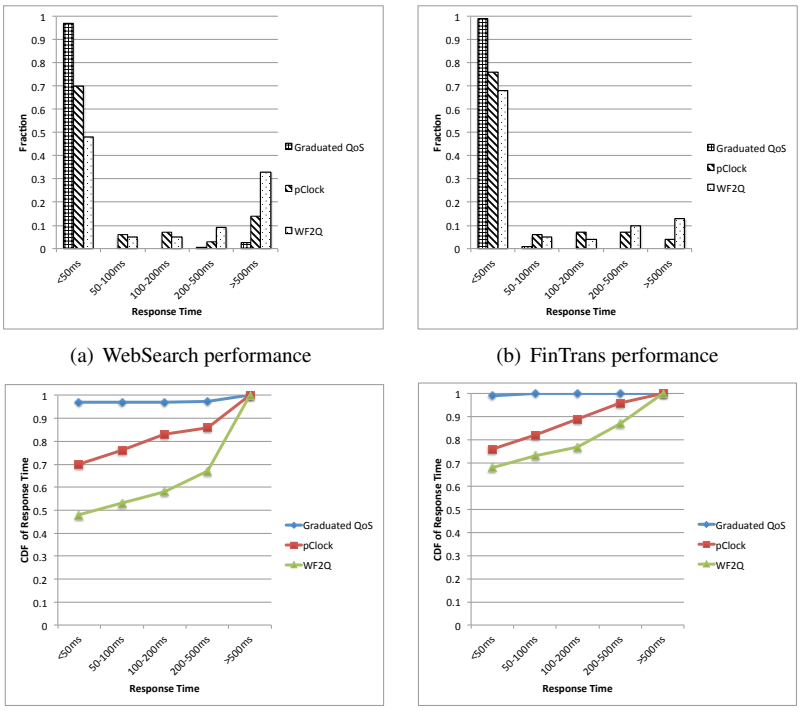
We end with an interesting case when the class parameters are multiples of the base value.

Lemma 2: Let $\alpha = \delta_{i+1}/\delta_i$, $\beta = \sigma_{i+1}/\sigma_i$ and $\lambda = \beta/\alpha$ be constants. The server capacity required to meet SLOs is no more than: $max_{1 \leq j \leq n}\{\rho,\ \lambda^j(\sigma_1/\delta_1) + \rho(1 - 1/\lambda^j)\}$. For $\lambda < 1$, the server capacity is bounded by $\sigma_1/\delta_1 + \rho$, which is less than twice the capacity required for servicing C_1.

6 Related Work

The simplest QoS model provides each client i a guaranteed server bandwidth of B_i IOPS. The server capacity is divided among the active clients in proportion to their guaranteed bandwidths, so that client i receives an allocation of $C \times w_i$, where C is the server capacity, $w_i = B_i / \sum_{j \in \mathcal{A}} B_j$ and $\mathcal{A}$ is the set of active clients. As long as the provisioned capacity C and the set of admitted clients $\mathcal{A}$ satisfy $C \geq \sum_{j \in \mathcal{A}} B_j$, the QoS guarantees for all clients can be met. A large number of algorithms have been proposed for proportional resource sharing *e.g.* Fair Queuing [8], WFQ [17, 5], WF2Q [3], Start Time Fair Queuing [7], Self-Clocking [6] etc. The general idea is to emulate the behavior of an ideal (continuous) Generalized Processor Sharing (GPS) scheduler in a discrete system, and divide the resource at a fine granularity in proportion to client weights. With proportional allocation, it is not possible to specify an independent response time requirement that is unrelated to its throughput. The WF2Q algorithm [3] guarantees that the worst-case response time of a request of client i is bounded by the time to serve all its queued requests at a uniform rate B_i without any additional delay. QoS models and algorithms for providing throughput guarantees when system capacity can vary, were presented in [11].

A second QoS model focuses on providing latency controls along with proportional sharing [17, 10]. In addition to providing minimum bandwidth guarantees, individual requests are guaranteed a maximum response time provided the client traffic satisfies stipulated constraints on burst size and arrival rate. Cruz *et al.* [17, 5] utilize the service curves concept to regulate workload patterns and arrival rates. They provide the SCED algorithm to schedule workloads specified by a given set of service curves. However, a major problem of the SCED algorithm is that it may result in starvation of a client which uses spare system capacity. Gulati *et al.* propose an algorithm *p*Clock [10], which uses a token bucket to control the arrival burst size and flow rate, and provide a synchronization scheme to avoid starvation; further, by setting the deadline of a request to be as late as possible the method attains greater flexibility in scheduling spare capacity. An issue not addressed by these methods is the impact on QoS guarantees of a badly behaved workload which violates its arrival constraints. Since these methods do not isolate the non-compliant part of the workload from its well-behaved portions, even small violations can lead to



(a) WebSearch performance

(b) FinTrans performance

(c) WebSearch: CDF of Response time

(d) FinTrans: CDF of Response time

Figure 7. Performance for Multiplexing

loss of QoS guarantees over extended (unbounded) portions of the workload. In addition, only a single response time guarantee is supported by this model, so the flexibility is limited and the provisioned capacity requirements are high.

More closely related to nested QoS are network QoS models where traffic shaping is used to decompose the workloads, and provide performance guarantees in terms of bandwidth and latency. Typically, arriving network traffic is made to conform to a token-bucket model by regulating the arrivals, and dropping requests that do not conform to the bucket parameters. With this drop-and-retransmission mechanism, the workload performance is guaranteed for the admitted portion of the workload, and the server utilization is maximized. However, drop-and-retransmission is not generally acceptable in storage systems, whose protocols do not support automatic retry.

The Nested QoS model classifies different portions of the workload into different classes and schedules them with different response time bounds. Empirical study of storage workloads to show the benefits of exempting a fraction of the workload from response time bounds was shown in [13], and used in the design of a slack-based two-level scheduler for a single client workload in [14]. However, there was no formal QoS model underlying the approach, that precluded specifying a well-defined SLO.

7 Conclusions and Future Work

The Nested QoS model provides several advantages over usual SLO specifications: (i) large reduction in server capacity without significant performance loss, (ii) accurate analytical estimation of the server capacity, (iii) providing flexible SLOs to clients with different performance/cost tradeoffs, and (iv) providing a clear conceptual structure of SLOs in workload decomposition. Our work continues to explore alternative implementations, capacity estimation for unrestricted parameters, relating workload characteristics with nested model parameters, semantic restrictions on decomposition, scheduling multiple decomposed workloads on a shared server, and a Linux block-level implementation.

Acknowledgements: We thank our shepherd Ali Mashtizadeh for his encouragement and help in improving the paper. Support by NSF Grants CNS 0615376 and CNS 0917157 is gratefully acknowledged.

References

[1] Storage Performance Council (UMass Trace Repository), 2007. http://traces.cs.umass.edu/index.php/Storage.

[2] SNIA: IOTTA Repository, 2009. http://iotta.snia.org.

[3] J. C. R. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *INFOCOM (1)*, pages 120–128, 1996.

[4] C.-S. Chang. *Performance Guarantees in Communication Networks*. Springer-Verlag, London, UK, 2000.

[5] R. L. Cruz. Quality of service guarantees in virtual circuit switched networks. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, 1995.

[6] S. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOMM'94*, pages 636–646, April 1994.

[7] 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.

[8] A. G. Greenberg and N. Madras. How fair is fair queuing. *J. ACM*, 39(3):568–598, 1992.

[9] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In *Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT '09)*, 2009.

[10] A. Gulati, A. Merchant, and P. Varman. *p*Clock: An arrival curve based approach for QoS in shared storage systems. In *ACM SIGMETRICS*, 2007.

[11] A. Gulati, A. Merchant, and P. Varman. *m*Clock: Handling Throughput Variability for Hypervisor IO Scheduling . In *USENIX OSDI*, 2010.

[12] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[13] L. Lu, K. Doshi, and P. Varman. Workload decomposition for qos in hosted storage services. In *MW4SOC*, 2008.

[14] L. Lu, K. Doshi, and P. Varman. Graduated QoS by decomposing bursts: Don't let the tail wag your server. In *29th IEEE International Conference on Distributed Computing Systems*, 2009.

[15] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron. Everest: Scaling down peak loads through i/o off-loading. In *Proceedings of OSDI*, 2008.

[16] K. I. Park. *QoS In Packet Networks*. Springer, USA, 2005.

[17] H. Sariowan, R. L. Cruz, and G. C. Polyzos. Scheduling for quality of service guarantees via service curves. In *Proceedings of the International Conference on Computer Communications and Networks*, pages 512–520, 1995.