USENIX Association

Proceedings of FAST '03: 2nd USENIX Conference on File and Storage Technologies

San Francisco, CA, USA
March 31–April 2, 2003

Façade: virtual storage devices with performance guarantees

Christopher R. Lumb* Arif Merchant† Guillermo A. Alvarez‡
Hewlett-Packard Laboratories

Abstract

High-end storage systems, such as those in large data centers, must service multiple independent workloads. Workloads often require predictable quality of service, despite the fact that they have to compete with other rapidly-changing workloads for access to common storage resources. We present a novel approach to providing performance guarantees in this highly-volatile scenario, in an efficient and cost-effective way. Façade, a virtual store controller, sits between hosts and storage devices in the network, and throttles individual I/O requests from multiple clients so that devices do not saturate. We implemented a prototype, and evaluated it using real workloads on an enterprise storage system. We also instantiated it to the particular case of emulating commercial disk arrays. Our results show that Façade satisfies performance objectives while making efficient use of the storage resources—even in the presence of of failures and bursty workloads with stringent performance requirements.

1 Introduction

Driven by rapidly increasing requirements, storage systems are getting larger and more complex than ever before. The availability of fast, switched storage fabrics and large disk arrays has enabled the creation of data centers comprising tens of large arrays with thousands of logical volumes and file systems, for total capacities of hundreds of terabytes and aggregate transfer rates of tens to hundreds of GB/s.

Such a consolidated data center typically serves the storage needs of the organization it belongs to, or even of multiple organizations. For example, companies that outsource their data storage and management contract the services of a Storage Service Provider (SSP); large organizations may also follow this model internally, to satisfy the requirements of separate divisions. The SSP allocates storage on its own disk arrays and makes it available to the customer over a network. Because the SSP serves multiple customers, it can do so more efficiently than a single customer—who may lack the space, time, money and expertise to build and maintain its own storage infrastructure. But this strength of the SSP can also be its bane: independent workloads compete for storage resources such as cache space, disk, arm, bus, and network bandwidth, and controller cycles.

A customer's contract with an SSP frequently includes a Service Level Agreement that combines a *Service Level Objective* (SLO) specifying the capacity, availability and performance requirements that the provided storage will meet, plus the financial incentives and penalties for meeting or failing to meet the SLO. We concentrate on the problem of providing performance guarantees. A primary requirement is *performance isolation:* the performance experienced by the workload from a given customer must not suffer because of variations on the workloads from other customers. Additional requirements may be imposed by customers moving from internally-managed storage to an SSP, or from

*Current address: Carnegie Mellon University, Hamerschlag Hall, Pittsburgh, PA 15213, USA, valheru@ece.cmu.edu

†Current address: 1501 Page Mill Rd., MS 1134, HP Laboratories, Palo Alto, CA 94304, USA. arif@hpl.hp.com

‡Current address: IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA., alvarezg@almaden.ibm.com

one SSP to another: for example, "My application runs well on an XP-1024 array; I want performance similar to this."

Traditional techniques for providing guarantees in the networking domain do not readily apply to storage, and adaptation at the application level is extremely rare [24]. The storage system is therefore left with the task of apportioning its resources with very little knowledge of the highly-variable workloads, and subject to constraints implied by protocols originally designed to provide best-effort service. One approach commonly followed to ensure performance isolation is to overprovision to the point that performance is no longer an issue. The resulting system can easily be twice as expensive as a correctly-designed system [1, 20]—a substantial difference for hardware and management costs in the order of millions of dollars. Another alternative is to assign separate physical resources (e.g., separate arrays) to different customers. This inflexible solution is impractical for a heterogeneous system that contains a time-evolving mix of newer and older devices, and makes it difficult to add capacity in arbitrary increments without extensive reconfiguration when customer requirements change. It can also cause overprovisioning by constraining the SSP to the design points of devices in the market: a given customer's workload may use a large fraction of a disk array's bandwidth, but only a small portion of its capacity. Besides, it does not solve the problem of resource allocation: the actual performance characteristics of real arrays are notoriously hard to model and predict.

We propose to solve this problem by adding one level of virtualization between hosts and storage devices. Façade provides the abstraction of virtual stores with performance guarantees, by intercepting every I/O request from hosts and redirecting them to the back-end devices. Unlike raw disk arrays (which do not guarantee predictable performance) Façade provides, by design, statistical guarantees for any client SLO. Virtual stores are not restricted by the design decisions embodied in any particular array, and can effectively shield client hosts from the effects of adding capacity or reconfiguring the back-end.

To evaluate this idea, we implemented a Façade prototype as a software layer between the workload-generator/trace-replayer and the storage device. Façade can also be implemented as a *shim box*, a thin controller that sits between the hosts and the storage. We subjected the prototype to a variety of workloads using several different commercial disk arrays s back-end devices. We then applied Façade's basic mechanism to the particular case of disk array virtualization: providing virtual stores that have the same performance characteristics as existing disk arrays. We found that Façade is able to satisfy stringent SLOs even in the presence of dynamic, bursty workloads, with a minimal impact on the efficiency of the storage device. The performance of Façade virtual storage devices is close to that of the device being emulated, even with failures on the host storage device.

Our experiments indicate that Façade can support a set of workloads with SLOs on a storage device if the device is capable of supporting them. We assume that there is an external capacity planning/admission control component which limits the workloads presented to Façade to what can be supported on the device. Façade can facilitate admission control by indicating when the device is approaching its limits; however, the design of an admission control component is outside the scope of this paper.

The remainder of this paper is as follows. Section 2 describes some related work. We introduce the architecture and internal policies of Façade in Section 3. We then evaluate Façade in Section 4, both on general SLOs and for array virtualization; we then conclude in Section 5.

2 Related work

There is an extensive body of work on networking SLAs [3] and network flow-control methods going back to the leaky bucket throttling algorithm [23] and fair queueing [18, 9]. There are several architectures for providing different levels of services to different flows, including the Differentiated Services (DiffServ) architecture [4] and the QoSBox [7], by mapping groups of flows with similar requirements into a few classes of traffic. Unfortunately, most of the techniques used do not gen-

eralize to shared storage systems [24]: dropping SCSI packets to relieve congestion is not an option, no traffic shapers exist within the system, the performance of storage devices is much more dependent on their current states than in the case of network components, and simple linear models are completely inadequate for performance prediction.

Wilkes suggested the notion of a storage QoS controller that can be given per-workload performance targets and enforces these using performance monitoring, feedback based rate control and traffic shaping.

Prior solutions have relied upon real-time scheduling techniques [11]. Work on multimedia servers [10] has primarily addressed the case in which multiple similar streams must be serviced; our version of the problem is more challenging, as we have to process an arbitrary mix of workloads, client hosts do not cooperate with our throttling scheme, and we cannot tolerate long startup delays when a workload arrives into the system. The YFQ disk scheduling algorithm [5] is an approximate version of generalized processor sharing that allows applications to reserve a fixed proportion of a disk's bandwidth. Sullivan used YFQ with lottery scheduling to manage disk resources in a framework that provides proportional share allocation to multiple resources [22]. While YFQ does isolate the performance offered to different applications, it does not support have the notion of per-stream average latency bounds as Façade does. The Cello framework [21] arbitrates disk resources among a heterogeneous mix of clients. It depends on correctly pigeonholing incoming requests into the a limited set of application classes (e.g., "throughput-intensive, best-effort"), on assigning the correct relative priorities to each class, and on being able to compute accurate estimated service times at the device. Façade is fully adaptive, can process a richer variety of clients, and does not depend on the existence of accurate device models.

The concept of storage virtualization, where strings of storage devices are logically coalesced, has existed for over two decades in the MVS mainframe operating system [8]. Modern operating systems include some form of logical volume manager (e.g., HP-UX LVM [17]), which stripe fault-tolerant logical volumes over multiple physical devices. LVMs do not provide performance guarantees and must be configured separately on each client host. A number of current commercial products provide storage virtualization across storage devices in a storage area network, including SANsymphony from DataCore Software [19], IPstore from FalconStor software [16] and the StorageApps sv3000 virtualization appliance [14]; none of them, however, provide performance guarantees for applications.

Automatic system design tools like Minerva [1] and Hippodrome [2] build systems that satisfy declarative, user-specified QoS requirements. They effectively minimize overprovisioning by taking into account workload correlations and detailed device capabilities to design device configuration and data layouts. The whole storage system may be redesigned in every refinement iteration, and there typically is a substantial delay to migrate the data online. As a result, they adapt to changes much more slowly than Façade. We view these tools as complementary to this work: Façade can handle short-term workload variations through adaptive scheduling without migrating data, and possibly postpone the need for a heavyweight system redesign.

3 Structure and components of Façade

Façade is a dynamic storage controller for controlling multiple I/O streams going to a shared storage device, and to ensure that each of the I/O streams receives a performance specified by its service-level-objective (SLO). Façade is designed for use in a storage management system (see Figure 1) with a separate capacity planning component. The capacity planner allocates storage for each workload on the storage device and ensures that the device has adequate capacity and bandwidth to meet the aggregate demands of the workloads assigned to it. This allocation may be changed periodically to meet changing workload requirements and device configurations [2]; however, such reallocation occurs on a time scale of hours to weeks. Façade manages device resources in time scale of milliseconds, to enable SLO compliant per-

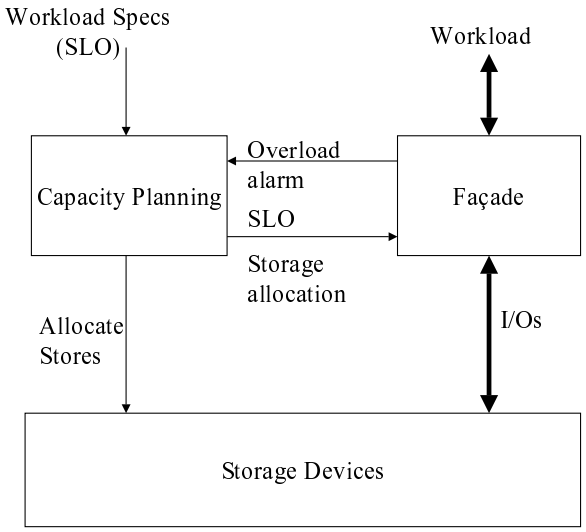


Figure 1. Façade controller in a storage management system.

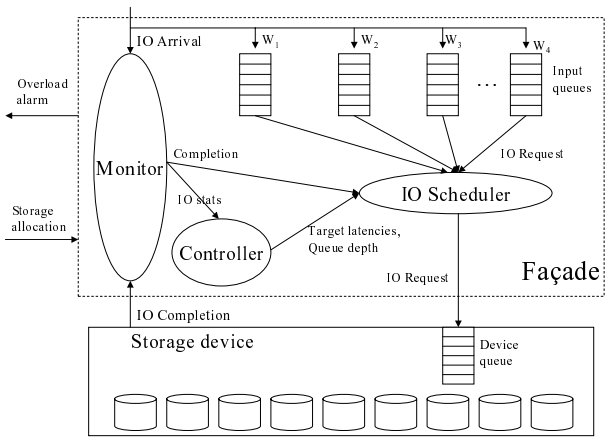


Figure 2. Façade architecture.

formance isolation between dynamically varying workloads.

Figure 1 shows the flow of information through the storage management system. Workload performance requirements, specified as an SLO, are given to the capacity planner (possibly by a system administrator). The capacity planner allocates storage on a storage device, possibly using models to determine if the device has adequate bandwidth to meet the requirements of the workload in addition to those already on it. It passes the allocation information and the SLO requirements to Façade. The workload sends I/O requests to Façade, which communicates with the storage device to complete the requests. If Façade is unable to meet the SLO requirements, this is communicated to the capacity planner, possibly triggering a re-allocation.

A workload SLO consists of a pair of curves specifying read and write latency as a function of the offered request rate, plus a time window length w. Average latency over a window should not exceed the weighted average of the specified latency bounds for the read/write mix and offered request rate during the window. Formally, we represent the two curves discretely as a vector of triples $((r_1,tr_1,tw_1),(r_2,tr_2,tw_2),\ldots,(r_n,tr_n,tw_n))$ where $0 < r_1 < \ldots < r_n$. We divide time into windows (epochs) of length w. For a workload with fraction of reads f_r, the average latency over any time window should not exceed $f_r tr_i + (1-f_r)tw_i$ if the offered request rate over the previous window is less than r_i. This formula implies a latency bound of tr_i for read only workloads, tw_i for write-only workloads, and a linear interpolation between the two bounds for mixed read/write workloads. By default, we assume $r_0 = 0$ and $r_{n+1} = tr_{n+1} = tw_{n+1} = \infty$: in other words, there is no bound on the average latency if the offered load exceeds r_n. For example, consider an application that generates up to 100 transactions per second, each of which causes up to 3 IOs. The transactions are required to complete in 100ms on the average. In this case, one might require an average latency of no more than 33ms per IO so long as the IO rate is no more than 300 IOs/sec, leading to the SLO $((300, 0.03, 0.03))$.

Façade structure

Façade implements SLOs through a combination of real-time scheduling and feedback-based control of the storage device queue (see Figure 2). This control is based on very simple assumptions about the storage device: we assume only that reducing the length of the device queue reduces the latency at the device, and increasing the device queue may increase throughput. These properties are satisfied by most disks and disk arrays.

Requests arriving at Façade are queued in per-workload input queues. Façade has three main

components that determine when these requests are sent to the underlying storage device: an earliest deadline first (EDF) IO scheduler, a statistics monitor that collects I/O statistics, and a controller that periodically adjusts targets for device queue depth and workload latency. We describe these in order.

IO scheduler

Based on periodic input from the controller, the IO scheduler maintains a target queue depth value and per-workload latency targets, which it tries to meet using Earliest Deadline First (EDF) scheduling. The deadline for a request from workload W_k is $arrivalTime(W_k) + latencyTarget(W_k)$, where $arrivalTime(W_k)$ is its arrival time and $latencyTarget(W_k)$ is a target supplied for W_k by the controller. The deadline for the workload W_k is the deadline of its oldest pending request.

The scheduler polls the device queue depth and the input queues periodically (every 1ms in our implementation) and also upon IO completions. Requests are admitted to into the device queue in two cases. (1) If the device queue depth is now less than the current queue length target (supplied by the controller — see ahead), the scheduler repeatedly picks the workload with the earliest deadline and sends the first request in its queue to the device until the device queue depth target is met. (2) If the deadline for any workload is already past, the past-due requests from that workload are scheduled even if this causes the queue depth target to be exceeded. Since the intent of controlling queue depth is to allow workloads with low latency requirements to satisfy their SLOs, it is not sensible to throttle these workloads: this rule allows newly-arrived low-latency workloads to be served even as the queue depth adapts.

Statistics monitor

The monitor receives IO arrivals and completions. It reports the completions to the IO scheduler, and also computes the average latency and read and write request arrival rates for active workloads every P seconds and reports them to the controller. This control period length P is a tuneable parameter; we used $P = 0.05$.

Controller

The controller periodically (every P seconds) adjusts the target workload latencies and the target device queue length. The target workload latencies must be adjusted because, as the workload request rates vary, Façade must give those requests a different latency based on the workload SLO. The device queue depth must also be adjusted to meet these varying workload requirements. The controller tries to keep the device queue as full as possible while still meeting latency targets, since a full device queue improves device utilization and the throughput it can produce. However, long device queues usually mean long latencies; hence, when any workload demands a low latency, the controller reduces the target queue depth. Reducing the queue size ensures that there are not too many outstanding I/Os in the device queue when an I/O requiring low latency arrives. When it arrives, it will be the next one to be sent to the device, and it will execute faster because the device has fewer outstanding I/Os.

The controller uses the IO statistics it receives from the monitor every P seconds to compute a new latency target based on the SLO for each workload as follows. Formally: suppose the SLO for workload W_k is $((r_1, tr_1, tw_1), (r_2, tr_2, tw_2), \ldots, (r_n, tr_n, tw_n))$ with a window w, and the fraction of reads reported is f_r. Let $r_0 = 0$, $r_{n+1} = \infty$, $tr_{n+1} = tw_{n+1} = \infty$. Then

$$latencyTarget(W_k) = tr_i f_r + tw_i(1 - f_r)$$
$$\text{if } r_{i-1} \leq readRate(W_k) + writeRate(W_k) < r_i.$$

The controller also computes a (possibly) new target queue depth for the storage device, based on the IO latencies measured in the previous control period and the current latency targets. If any workload has a new target latency lower than that measured in the previous control period, the queue depth target is reduced proportionately. On the other hand, if the new target latencies are all larger than those achieved in the previous control period, then the queue depth target can be increased. We check if the queue depth in the previous control period was in fact limited by the queue depth target; if so, we raise the new queue depth target slightly

to allow for greater throughput. Formally, this is a non-linear feedback controller implemented by the following equations: Suppose the measured average IO latency for workload W_k is $L(W_k)$, the maximum queue depth achieved in the last control period is Q_{max} and the previous target device queue depth is Q_{old}. Then, the new queue depth target Q_{new} is computed as follows:

$$E = \min_k \frac{latencyTarget(W_k)}{L(W_k)}$$

$$Q_{new} = \begin{cases} E \cdot Q_{old} & \text{if } E < 1, \\ (1+\varepsilon)Q_{old} & \text{else if } Q_{max} = Q_{old}, \\ Q_{old} & \text{otherwise.} \end{cases}$$

Here, ε is a small positive value; we use $\varepsilon = 0.1$. The intial queue depth is set to 200 entries.

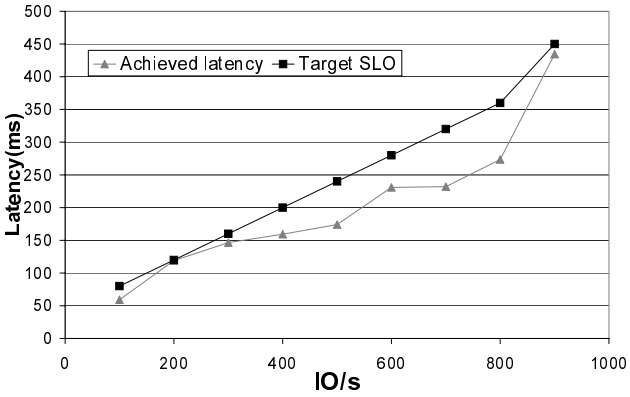


Figure 3. Latency at various IO rates is compared with SLO for a Façade VSD on a FC-60 LU.

4 Experimental evaluation

We empirically validated the accuracy, stability and efficiency of our techniques for providing virtual stores with QoS guarantees through experiments on two modern commercial arrays. The FC-60 [12] used in our experiments is a mid-range array with 30 Seagate Cheetah ST118202LC disks (10,000rpm, 18.21GB each), and two controllers with 256MB of NVRAM cache in each. We set up a RAID5 Logical Unit (LU) using 6 disks on this array. The VA-7100 [15] is a small array with 10 Seagate Cheetah ST318451FC disks (15000rpm, 18.35GB each) and two controllers each with 256MB of NVRAM cache. We configured two AutoRAID [25] LUs on this. Each array is connected to an HP 9000-N4000 server through a Brocade Silkworm 2800 switch using two FibreChannel links.

We employed both synthetic and trace workloads in this evaluation, using the Buttress workload generator, which can produce synthetic workloads and replay traces. We used synthetic traces consisting of 67% reads and 33% writes, distributed randomly over the LU; the I/O rates are described with the experiments and the workloads were asynchronous (open) except where specified. The trace workload is an I/O trace from a production OpenMail email server [13]. For each workload execution, we collected traces of I/O activity at the device driver level, including I/O submission and completion times, read/write characteristics, size and logical address information. These traces were then analyzed to provide throughput, latency and other statistics.

We implemented Façade as a software layer between Buttress and the device driver. A Virtual Storage Device (VSD) is a virtual LU provided by the Façade layer. It has an associated Service Level Objective (SLO) and stores its data on a real (physical) LU. One or more VSDs may use the same underlying real LU as a backend device.

We now describe a series of experiments designed to test how well Façade meets its goals: to match the SLO, to provide performance isolation between workloads, to emulate LUs on a different disk array, and to operate without significantly reducing the overall efficiency of the underlying hardware. In each case, the SLO window size is 1 second.

4.1 SLO compliance

This experiment tests how well Façade ensures compliance with a SLO. A Façade VSD was created on an FC-60 LU, with a specified SLO. We then ran a synthetic workload with 67% reads and 33% writes, increasing the IO rate in increments of 50 IOs/sec against the VSD. There is an additional background workload with a high IO re-

quest rate, but no latency requirement. Figure 3 is a throughput-latency plot showing the measured latency of each throughput range for the Façade VSD and the target based on the SLO. As the graph shows, the Façade VSD latency remains slightly lower than the latency bound in the SLO.

4.2 Performance isolation

An essential property that Façade should satisfy is performance isolation: workloads on different VSDs should not interfere with each other. In particular, the performance of the workload on one Façade VSD should not be fall below its SLO due to bursts in the load on another Façade VSD sharing the same physical hardware. We verify here how well our implementation provides this property.

Figure 4 shows time-plots of latency and throughput for two Façade VSDs, both using same FC-60 LU as a back-end device. VSD1 has a bursty workload: 200 IOs/sec, repeatedly on for 5 seconds and off for 5 seconds. VSD1 also has a relatively tight latency bound of 50ms: the corresponding SLO is $(200\text{IOs}/\text{sec}, 50\text{ms}, 50\text{ms})$. VSD2 has a stable workload of 500 IOs/sec, but a much looser latency requirement of 4000ms; the corresponding SLO is $((500\text{IOs}/\text{sec}, 4000\text{ms}, 4000\text{ms}))$. Both workloads are 4KB IOs, 67% reads and 33% writes.

The latency time-plot shows that VSD1 receives the latency it requires—while on, its average latency is approximately 50ms. The latency and throughput seen by VSD2 varies to comply with the requirements of VSD1; however, the requirements of the VSD2 SLO are also met. In particular, the latency provided by VSD2 is affected at approximately the same rate of change when a burst begins (increasing) and when it ends (decreasing). This is because the bursty workload has the more stringent latency requirement, so requests for the other workload are delayed in Façade, and spread evenly over a longer period of time; in fact, some VSD2 requests are delayed almost until the following burst starts. For VSD2, throughput varies between 300 and 800 IOs/sec, and the latency goes as high as 2000ms. When the same workloads are run on the FC-60 LU without Façade control, they are clearly not isolated from one another, as shown in Figure 4(a). The latency for VSD1 exceeds its SLO latency ($50ms$) regularly.

Workload	Without Façade	With Façade
VSD1	75 ms	42 ms
VSD2	72 ms	43 ms
VSD3	71 ms	3922 ms

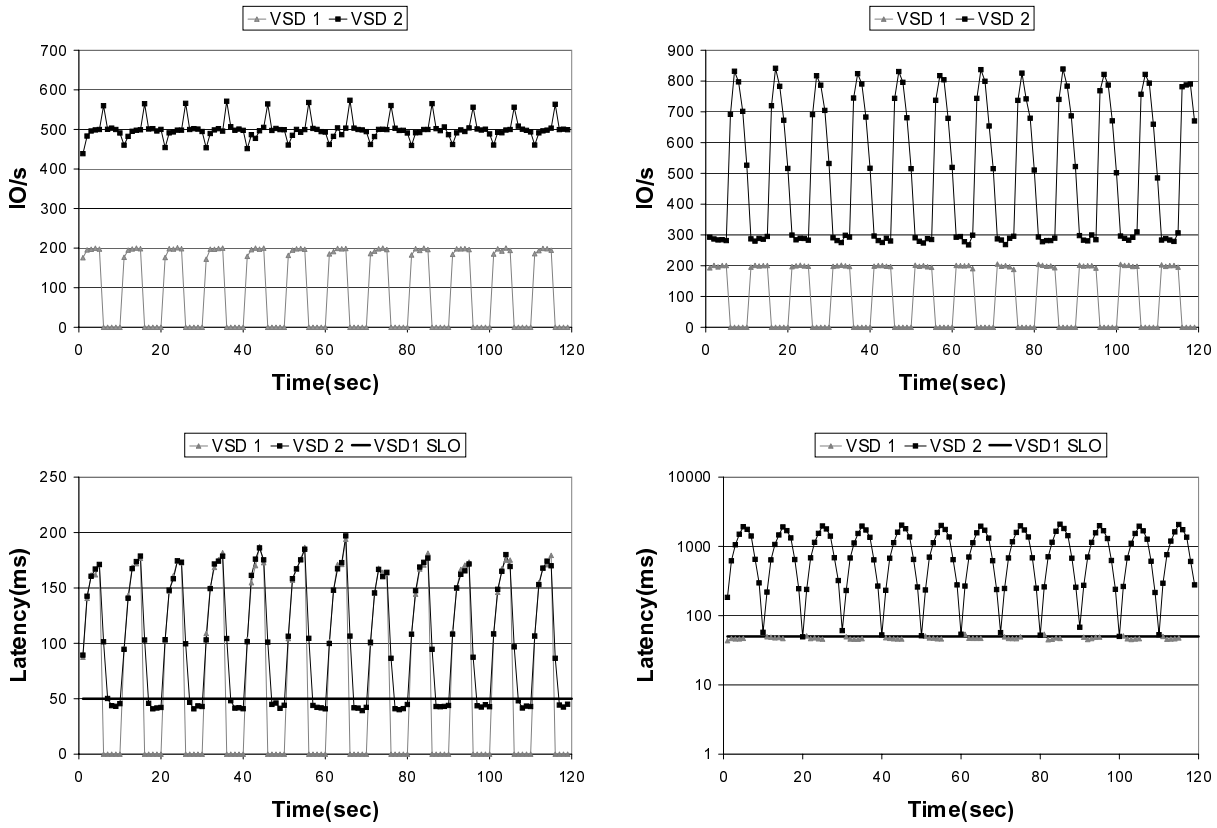
Table 1. Performance isolation experiment 2: Standard deviation of latency

Figure 5 shows an experiment with a more complex set of workloads. In this case we have three workloads, VSD1, VSD2, and VSD3. VSD1 is a workload running for 30s on, 10s off at 50 IO/s with a latency target of 55ms. VSD2 is a workload runs for 10s at 75 IO/s, 10s at 25 IO/s, 10s at 75 IO/s and then off for 10s. When VSD2 is issuing 25 IO/s its latency target is 30ms when it is issuing 50 IO/s its latency target is 60ms. VSD3 issues requests continuously and has a high latency target of 2000ms. Even with this complex set of workloads, where the SLO latency target of VSD1 is sometimes higher and sometimes lower than that of VSD2, Façade is able to track and meet all latency targets. Without Façade control, the latencies for VSD1, VSD2 and VSD3 are always similar and both VSD1 and VSD2 regularly miss their latency targets.

For this experiment, we also measured the standard deviation of latency, shown in Table 1. The standard deviation of latency for VSD1 and VSD2, the workloads with tight latency requirements, dropped by a factor of $1.67 - -1.78$ when Façade control was used; the standard deviation of the greedy workload VSD3 increased much. This is consistent with what one would expect intuitively: since Façade controls the latencies that the workloads receive, workloads with low latency requirements see latencies close to their requirement, and therefore a lower variance. Workloads with a high latency tolerance see sometimes low and sometimes high latencies, depending on load, and thus a higher variance.

4.3 Maximum SLO

The performance of a workload running on a dedicated LU gives an upper bound on the SLO that



(a) Without Façade control

(b) With Façade control

Figure 4. Performance isolation experiment 1.

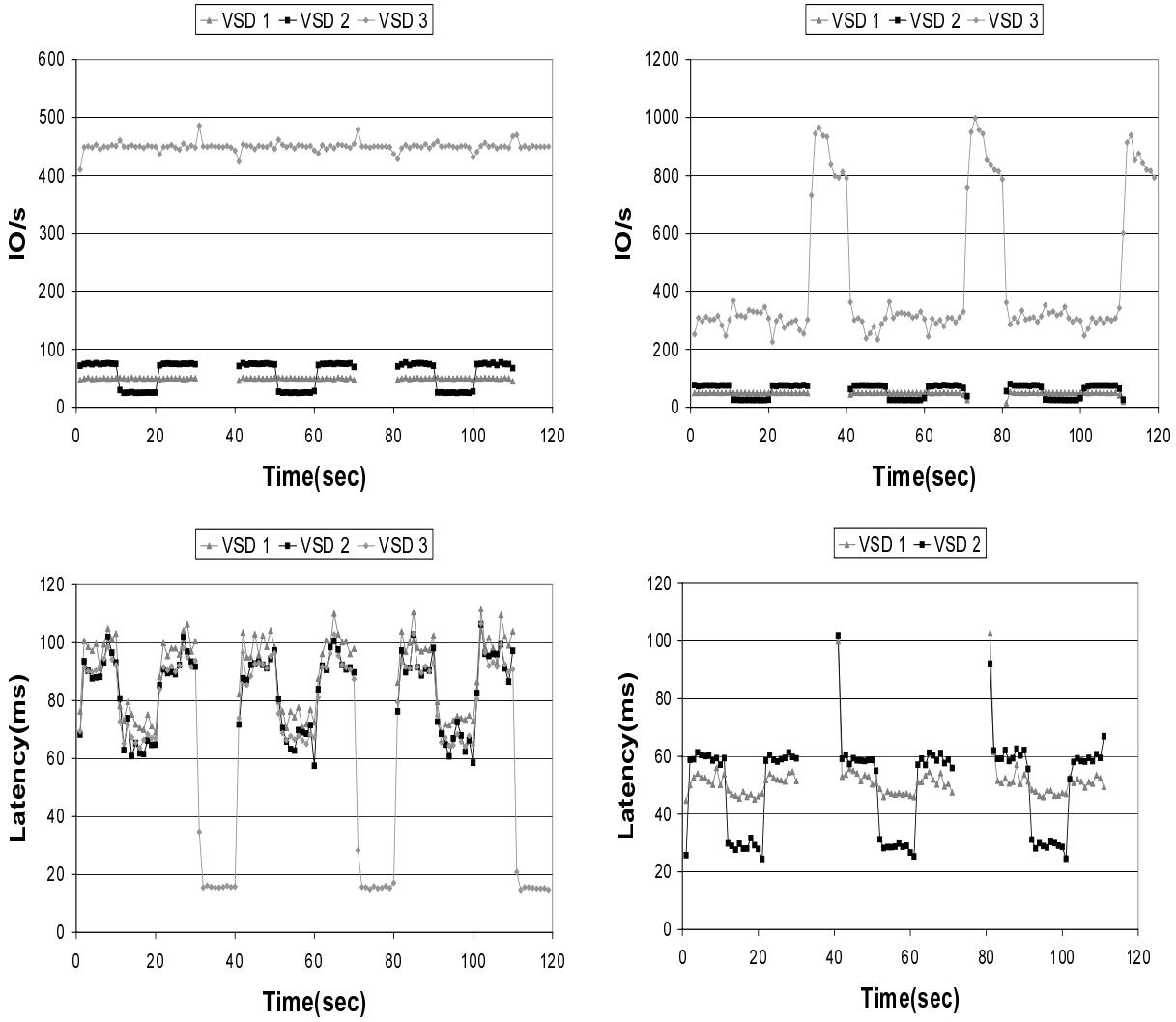
LU can support for that workload. We use this to test how stringent an SLO Façade can support. We measured the latencies at various IO rates on an otherwise unloaded FC-60 LU using our standard synthetic workload. We used this latency-throughput curve as the SLO for a Façade VSD, VSD0, running on the FC-60 LU; this should be the most stringent SLO the FC-60 LU can support for this workload. The workload for VSD0 has a gradually increasing IO rate. We also added a greedy workload with no specific latency requirement, VSD1, that issues IOs as quickly as possible to add load to the LU. More precisely, VSD1 is a synchronous (closed) workload with 2000 outstanding IOs (4KB random reads) and zero think-time.

Figure 6 shows the measured latency and throughput for VSD0 and VSD1. VSD0 tracks quite faithfully the SLO, which is the performance of the workload on a dedicated array and VSD1 is completely starved out after 80 seconds. This indicates that Façade can support the most stringent SLO that the underlying device can support.

4.4 Multiplexing

While performance isolation can be provided conventionally to workloads by segregating their data on LUs residing on separate hardware components, there can be an overall loss of performance as a result. (Equivalently, it may be necessary to overprovision the storage for each workload in order to meet performance requirements.) We show in this experiment that combining workloads on the same hardware using Façade can provide substantially better performance than segregating the workloads.

There are two workloads: VSD1 is a bursty workload that repeatedly requests 300 IOs/sec for



(a) Without Façade control

(b) With Façade control

Figure 5. Performance isolation experiment 2.
Latencies for VSD3 have been dropped from Figure 5(b) for clarity.

10 seconds, and then is off for 20 seconds; when on, it has a latency requirement of 20ms. VSD2 is a greedy workload (representing, for example, a backup); it has no specific latency requirement but will read data as quickly as possible. Figure 7 shows time plots of latency and throughput for VSD1 and VSD2 for two cases: (1) baseline: VSD1 and VSD2 each on a separate FC-60 LU and (2) VSD1 and VSD2 with their data striped across both LUs, using Façade. Our results show that when VSD1 and VSD2 run on separate LUs, VSD2 receives a steady throughput of approximately 847 IOs/sec. When VSD1 and VSD2 use both LUs under Façade, the throughput received by VSD2 is more variable, generally ranging between 450–1500 IOs/sec; however, the average throughput is 1146 IOs/sec, substantially higher than the throughput using separate LUs. VSD1 receives its required throughput and SLO latency in both cases; however, there is a spike in the latency whenever a

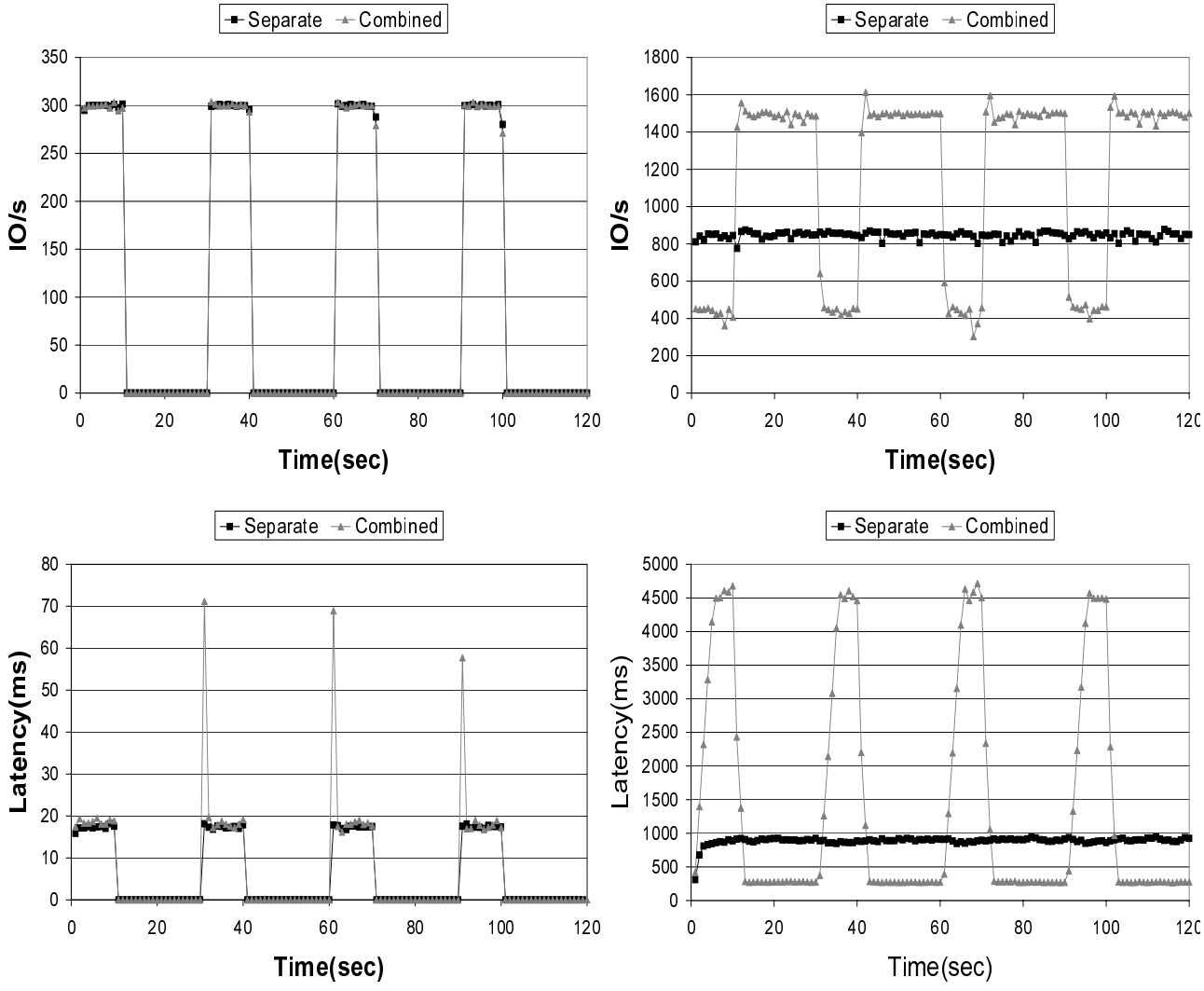


Figure 7. Combining VSDs on single hardware gives higher throughput to VSD2

burst of requests begins. The spike subsides quickly — within one time window. The spike occurs because Façade builds up a large device queue when VSD1 is off, in order to maximize the throughput of VSD2. When VSD1 comes on, the controller quickly reduces the device queue target, but it takes some time for the device queue to drain sufficiently that the VSD1 SLO latency target can be met.

4.5 Resource utilization

To test how efficiently Façade uses resources, we measured the size of the array required to support a workload, with and without Façade. We used the same workloads as in the first performance isolation experiment: VSD1, a synthetic workload of 200 IOs/sec, repeatedly on for 5 seconds and off for 5 seconds, with a latency target of 50ms when on, and VSD2, which requests 500 IOs/sec with a latency bound of 4000ms. We run two workloads together against a series of FC-60 RAID5 LUs with

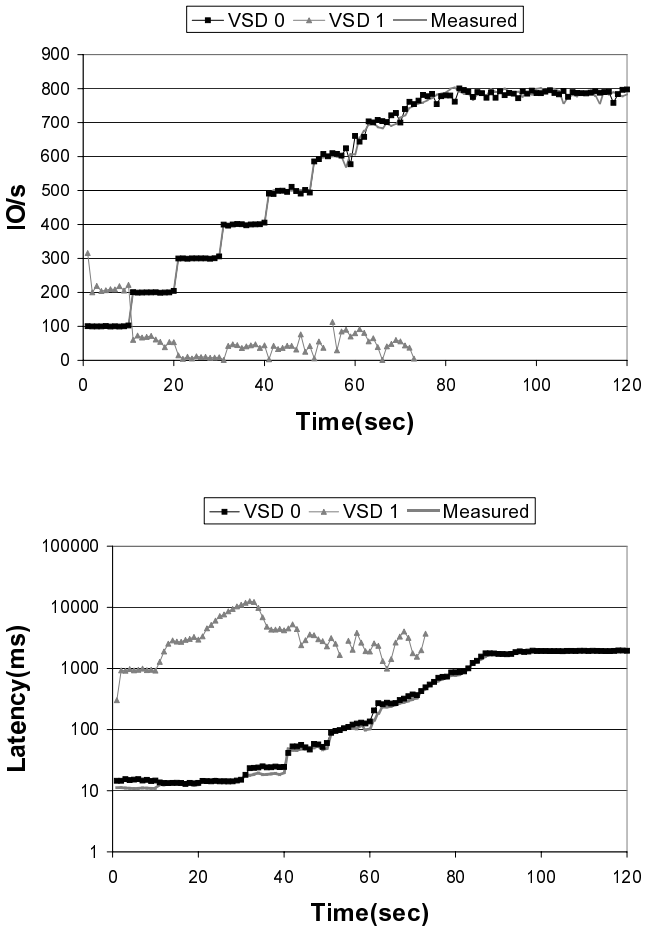


Figure 6. Façade matches maximum (dedicated) array performance. The "measured" data is VSD0 on a dedicated array without Façade, VSD0 and VSD1 are together with Façade control. Measured and VSD0 have complete overlap.

increasing numbers of disks, starting with 3 disks, until the latency target was met, with and without the use of Façade.

Figure 8 shows time plots of latency for VSD1 both with and without Façade. This figure shows that without Façade, 8 disks are required to fully meet the latency targets. With 7 disks the targets are missed by 10-20% and with 6 disks the latencies are almost double the target values.

Using Façade, considerably fewer resources are required to meet the target latencies. With only 3 disks Façade is usually able to meet the latency requirement; however, when a burst of requests starts, there is a latency spike similar to those seen in Section 4.4, due to the time required for draining the device queue. With 4 disks, there are sufficient resources in the system to drain the queue quickly, and the latency spikes disappear. Using Façade, the SLO latency can be met with 50% fewer resources.

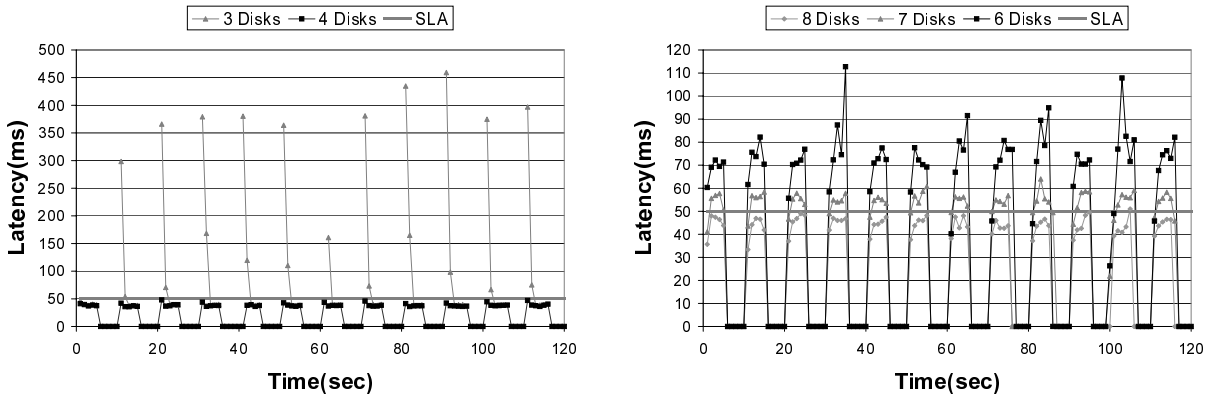
4.6 Façade overhead

Since Façade adds an additional control layer between the application and the storage device that modulates the queue depth at the storage device, it is possible that there is some loss of efficiency at the storage device. We measure the Façade overhead in this experiment.

For this experiment, we emulated a VA-7100 LU on itself, as follows. We measured the mean latency for reads and writes for a VA-7100 LU over a range of offered loads, and used this curve as an SLO. We formed a Façade VSD using this SLO and the same LU as the underlying device. We then replayed an Openmail trace against this VSD and (separately) directly against the underlying VA-7100 LU, measuring the throughput and latency over time for both. (A real trace was used for this test because the SLO was formed from measurements based on synthetic workloads.) Since the only difference between the two cases is the use of a Façade layer, the overhead due to Façade should appear as a loss of performance for the VSD run as compared to the run against the VA-7100 LU. Figure 9 shows the time plots of latency and throughput for both cases. It is clear that the performance of the LU under this workload is mostly unaffected by adding the Façade layer: overall, adding the Façade layer left throughput unchanged, and increased the average latency by less than 2%. We conclude that Façade adds a negligible overhead.

4.7 Performance under failure

When workloads are segregated on separate disk-groups in order to provide performance isolation, the impact of a disk failure on a workload can be dramatic. When multiple workloads are consolidated onto a shared group of disks, a single work-



With Façade Without Façade

Figure 8. VSD1 latency with and without Façade: the workload requires 8 disks to meet its target without Façade and only 4 disks with Facade.

load need not bear the entire brunt of the disk failure: instead, the effect can be reduced by spreading the performance loss over several workloads or shifted to workloads that can better tolerate the impact. Additionally, if the throughput of the shared disks is not completely used in normal mode then the impact of the failure can be reduced yet farther. We demonstrate this in the following experiment.

We measured the throughput and latency of replaying an Openmail trace in four configurations: (1) on a VA-7100 LU (without Façade control); (2) on a VA-7100 LU with one disk failed; (3) simultaneously, with small time offsets, on 2 Façade VSDs on an FC-60, each configured with SLOs to match the performance characteristics of a VA-7100 LU; and (4) in the same configuration as (3), but with one disk failed on the FC-60. On the FC-60 configurations, there is an additional background load of 450 IOs/sec with no latency requirement.

Figure 10 shows time plots of latency and throughput for one trace-replay for configurations (1), (2) and (4). The VA-7100 LU plot serves as a baseline for comparison and as the SLO for the Façade plot. For the uncontrolled VA-7100 LU, the latency increases by two orders of magnitude when a disk fails, and the throughput drops by more than one order of magnitude compared with the normal-mode VA-7100 baseline. We have not shown plots for the Façade VSD on the FC-60 in normal mode (configuration 3) to avoid cluttering the figures; however, these would show a performance similar to the normal-mode VA-7100. The plots for the Façade VSD on the FC-60 with one failed disk also show a performance almost identical to the VA-7100 normal mode performance. The effects of the disk failure are entirely absorbed by the background workload (not shown), which suffers a 10% drop in throughput compared with the normal-mode FC-60 operation.

We have demonstrated in Section 4.1 that Façade faithfully matches the specified SLO when the array it is running on is operating normally. This experiment shows further that, if possible, the SLO will be matched also in degraded mode.

5 Conclusions and future work

Traditional solutions to the problem of providing guaranteed performance to independent, competing clients are either inaccurate or substantially more expensive than necessary. By virtue of its fine-grain monitoring and decision-making, Façade is able to use resources much more efficiently, and to balance the load among multiple back-end devices while satisfying the performance requirements of many different client applications. Our experiments, run on a representative mid-range storage system with

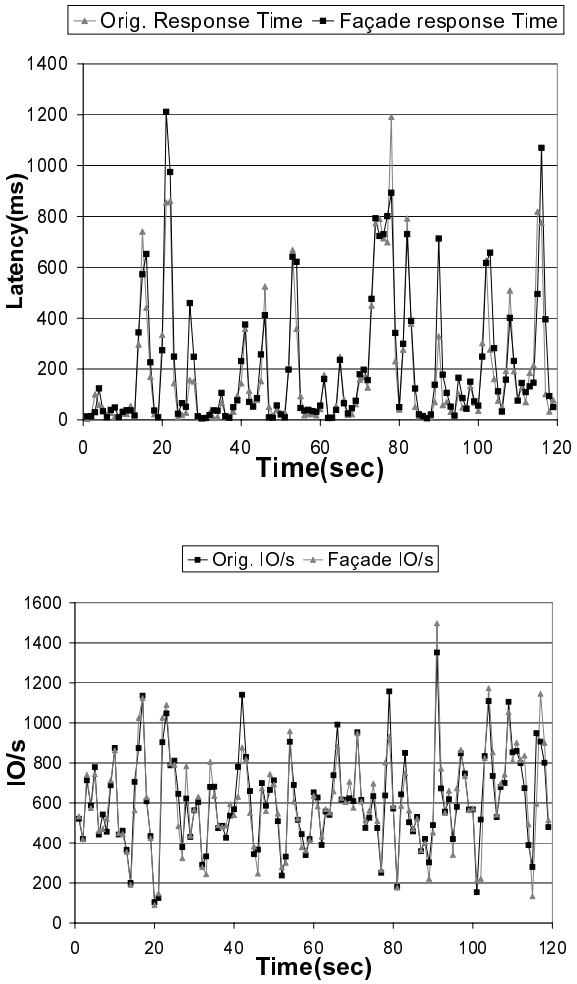


Figure 9. Façade overhead.

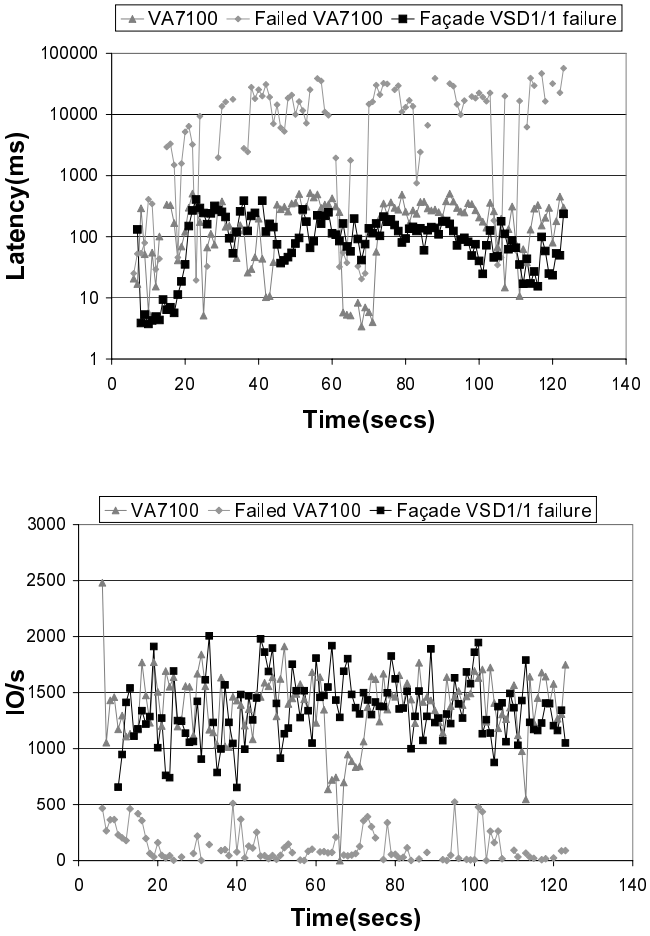


Figure 10. Latency and throughput for Openmail trace on VA7100 LU, VA 7100 LU with one disk failed, and Façade VSD on FC-60 with one disk failed.

commercial disk arrays, show that Façade meets SLOs with a very high probability while making efficient use of the storage resources. Façade can significantly reduce the hardware resources required to support a combination of workloads with tight latency requirements. Façade is capable of supporting workloads with performance requirements that change over time and it can handle as stringent an SLO as the underlying device can support. The performance penalty introduced by the additional processing layer is also negligible, and far outweighed by the resulting advantages. We believe that fully-adaptive storage virtualization appliances such as Façade should play a fundamental role in storage system design; not only does Façade provide fine-grain QoS enforcement, but it also adapts very quickly to changes in the workload, and is not strongly dependent on accurate workload characterizations or storage device models in order to function.

This work has postulated the existence of an admission control component. In future work, we hope to demonstrate the use of Façade to facilitate an adaptive admission control mechanism. Another potential extension for this work is to allow each client application to specify an elasticity coefficient [6] (i.e., a measure of how tolerant it is to deviations from the SLO) and exploit that flexibility in the controller algorithm. Also, multiple in-

stances of Façade in the same storage system could cooperate, in order to handle larger workloads that hit common back-end devices. We would also like to explore the implications of providing not only performance SLOs, but also availability/reliability guarantees, by dynamically changing the data layout in a way that is transparent to the client hosts.

Acknowledgements

We thank our shepherd, Eran Gabber, the anonymous reviewers and the members of the HP Labs Storage Systems Department for helpful comments on this work. In particular, we thank John Wilkes for suggestions on the choice of SLOs. We are also grateful to Hernan Laffitte for help with repeated disk array reconfigurations.

References

[1] G. A. Alvarez, E. Borowsky, S. Go, T. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes. Minerva: an automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems*, 19(4):483–518, November 2001.

[2] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administrators. In *Conference on File and Storage Technologies (FAST), (Monterey, CA)*, pages 175–188. USENIX, January 2002.

[3] C. Aurrecoechea, A. Campbell, and L. Hauw. A survey of QoS architectures. *Multimedia Systems*, 6(3):138–151, 1998.

[4] S. Blake et al. An architecture for differentiated services, December 1998. IETF RFC 2475.

[5] John L. Bruno, Jose Carlos Brustoloni, Eran Gabber, Banu Ozden, and Abraham Silberschatz. Disk scheduling with quality of service guarantees. In *ICMCS, Vol. 2*, pages 400–405, 1999.

[6] G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-time Systems*, 24(1–2):7–24, July 2002.

[7] N. Christin and J. Liebeherr. The QoSbox: A PC-router for quantitative service differentiation in IP networks. Technical Report CS-2001-28, University of Virginia, November 2001.

[8] IBM Corp. *MVS/DFP V3R3 System Programming Reference*, 1996. SC26-4567-02.

[9] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM Symposium on Communications Architectures and Protocols (Austin, Texas)*, pages 1–12, Sept 1989.

[10] J. Gemmell, H. Vin, D. Kandlur, V. Rangan, and L. Rowe. Multimedia storage servers: A tutorial. *IEEE Computer*, 28(5):40–49, 1995.

[11] P. Goyal and H. Vin. Generalized guaranteed rate scheduling algorithms: a framework. *IEEE/ACM Transactions on Networking*, 5(4):561–571, 1997.

[12] Hewlett-Packard Company, Palo Alto, CA. *HP SureStore E Disk Array FC60 User's Guide*, 2000. Pub. No. A5277-90001.

[13] Hewlett-Packard Company, Palo Alto, CA. *OpenMail Technical Reference Guide*, 2.0 edition, 2001. Part No. B2280-90064.

[14] Hewlett-Packard Company, Palo Alto, CA. *HP StorageApps sv3000 White Paper*, 2002.

[15] Hewlett-Packard Company, Palo Alto, CA. *HP StorageWorks Virtual Arrays, VA7000 Family, User and Service Guide*, 2002. Pub. No. A6183-96004.

[16] FalconStor Inc. Ipstor: Build an end-to-end IP-based network storage infrastructure. White paper. http://www.falconstor.com, 2001.

[17] T. Madell. *Disk and File Management Tasks on HP-UX*. Prentice Hall, Upper Saddle River, NJ, 1996.

[18] J. Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, 35(4):435–438, 1987.

[19] SANSymphony version 5 datasheet. http://www.datacore.com, 2002.

[20] G. Schreck. Making storage organic. Technical report, Forrester Research, May 2002.

[21] P. Shenoy and H. Vin. Cello: A disk scheduling framework for next generation operating systems. *Real-Time Systems*, 22(1-2):9–48, January 2002.

[22] David G. Sullivan and Margo I. Seltzer. Isolation with flexibility: A resource management framework for central servers. In *Proc. of the USENIX Annual Technical Conference (San Diego, California)*, pages 337–350, 2000.

[23] J. Turner. New directions in communications. *IEEE Communications*, 24(10):8–15, October 1986.

[24] J. Wilkes. Traveling to Rome: QoS specifications for automated storage system management. In *International Workshop on Quality of Service (Karlsruhe, Germany)*, pages 75–91, June 2001.

[25] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 96–108, Copper Mountain, CO, December 1995. ACM Press.