USENIX Association

Proceedings of the
FAST 2002 Conference on
File and Storage Technologies

Monterey, California, USA
January 28-30, 2002

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Aqueduct: online data migration
with performance guarantees

Chenyang Lu
Department of Computer Science
University of Virginia
chenyang@cs.virginia.edu

Guillermo A. Alvarez   John Wilkes
Storage and Content Distribution Department
Hewlett-Packard Laboratories
{galvarez, wilkes}@hpl.hp.com

Abstract

Modern computer systems are expected to be up continuously: even planned downtime to accomplish system reconfiguration is becoming unacceptable, so more and more changes are having to be made to "live" systems that are running production workloads. One of those changes is data migration: moving data from one storage device to another for load balancing, system expansion, failure recovery, or a myriad of other reasons. Traditional methods for achieving this either require application down-time, or severely impact the performance of foreground applications – neither a good outcome when performance predictability is almost as important as raw speed. Our solution to this problem, Aqueduct, uses a control-theoretical approach to statistically guarantee a bound on the amount of impact on foreground work during a data migration, while still accomplishing the data migration in as short a time as possible. The result is better quality of service for the end users, less stress for the system administrators, and systems that can be adapted more readily to meet changing demands.

1. Introduction

Current enterprise computing systems store tens of terabytes of active, online data in dozens to hundreds of disk arrays, interconnected by storage area networks (*SANs*) such as Fibre Channel [4] or Gigabit Ethernet [1]. Keeping such systems operating in the face of changing access patterns (whether gradual, seasonal, or unforeseen), new applications, equipment failures, new resources, and the needs to balance loads to achieve acceptable performance requires that data be moved, or *migrated*, between storage system components – sometimes on short notice. (We note in passing that creating and restoring online backups can be viewed as a particular case of data migration in which the source copy is not erased.)

Existing approaches to data migration either take the data offline while it is moved, or allow the I/O resource consumption engendered by the migration process itself to interfere with foreground application accesses and slow them down – sometimes to unacceptable levels. The former is clearly undesirable in today's global, always-on Internet environment, where people from around the globe are accessing data day and night. The latter is almost as bad, given that the predictability of information-access applications is almost as much a prerequisite for the success of a modern business as is their raw performance. We believe there is a better way; this paper explores our approach to the problem of performing general, online data migrations while maintaining performance guarantees for foreground loads.

1.1. Problem formulation

We formalize the problem as follows. The data to be migrated is accessed by *client applications* that continue to execute in the foreground in parallel with the migration. The inputs to the migration engine are (1) a *migration plan*, a sequence of data moves to rearrange the data placement on the system from an initial state to the desired final state [3], and (2) client application quality-of-service (QoS) demands – I/O performance specifications that must be met while migration takes place. Highly variable service times in storage systems (e.g., due to unpredictable positioning delays, caching, and I/O request reordering) and workload fluctuations on arbitrary time scales [10] make it difficult to provide absolute guarantees, so statistical guarantees are preferable unless gross over-provisioning can be tolerated.

The data migration problem is to complete the data migration in the shortest possible time that is compatible with maintaining the QoS goals.

1.2. QoS contracts

One of the keys to the problem is a useful formalization of the QoS goals. We use the following. A *store* is a logically contiguous array of bytes, such as a database table or a file system; its size is typically measured in gigabytes. Stores are accessed by *streams*, which represent I/O access patterns; each store may have one or multiple streams. The granularity of a stream is somewhat at the whim of the definer, but usually corresponds to some recognizable entity such as an application.

Global QoS guarantees bound the aggregate performance of I/Os from all client applications in the system, but do not guarantee the performance of any individual store or application. They are seldom sufficient for realistic application mixes, for access demands on different stores may be significantly different during migration (as shown in Sections 5 and 6). On the other hand, *stream-level guarantees* have the opposite difficulty: they can proliferate without bound, and so run the risk of scaling poorly due to management overhead.

An intermediate level, and the one adopted by Aqueduct, is to provide *store-level guarantees*. (In practice, this has similar effects to stream-level guarantees for our real-life workloads because the data-gathering system we use to generate workload characterizations creates one stream for each store by default.) Let the *average latency AL_i* of a store i in the workload be the average latency of I/Os directed to store i by client applications throughout the execution of a migration plan, and let the *latency contract* for store i be denoted LC_i. The latency contract is expressed as a *bounded average latency:* it requires that $AL_i \leq LC_i$ for every store i.

In practice, such QoS contract specifications may be derived from application requirements (e.g., based on the timing constraints and buffer size of a media-streaming server), or specified by hand, or empirically derived from workload monitoring and measurements.

We also monitor how often the latency bounds are violated over shorter time intervals than the entire migration, by dividing the migration into equal-sized *sampling periods*, each of duration W. Let M be the number of such periods needed for a given migration. Let the *sampled latency $L_i(k)$* of store i be its the average latency in the kth sampling period, which covers the time interval $((k-1)W, kW)$ since the start of the migration. We define the *violation fraction VR_i* as the fraction of sampling periods in which QoS contract violations occur: $VR_i = |\{k : L_i(k) > LC_i, k = 1,...,M\}| / M$.

1.3. Summary

The main contribution of the work we report here is a novel, control-theoretic approach to achieving these requirements. Our tool, *Aqueduct*, adaptively tries to consume as much as possible of the available system resources left unused by client applications while statistically avoiding QoS violations. It does so by dynamically adjusting the speed of data migration to maintain the desired QoS goals while maximizing the achieved data migration rate, using periodic measurements of the storage system's performance as perceived by the client applications. It guarantees that the average I/O latency throughout the execution of a migration will be bounded by a pre-specified QoS contract. If desired, it could be extended straightforwardly to provide a bound on the number of sampling periods during which the QoS contract was violated – but we found that it did so reasonably effectively without explicitly including this requirement, and suspected that doing so would reduce the data migration rate achieved – possibly more than was beneficial.

The focus in this paper is on providing latency guarantees because (1) our early work showed that bounds on latency are considerably harder to enforce than bounds on throughput – so a technique that could bound latency would have little difficulty with throughput; and (2) the primary beneficiaries of QoS guarantees are customer-facing applications, for which latency is a primary criterion.

To test Aqueduct, we ran a series of experiments on a state-of-the-art disk array [15] connected to a high-end host by a FibreChannel storage area network. Aqueduct successfully decreased the average I/O latencies of our client application workloads by as much as 76% compared with non-adaptive migration methods, successfully enforced the QoS contracts, and migrated data at close to the maximum speed allowed by the QoS guarantees. Although the current Aqueduct prototype does not explicitly guarantee a bound on the violation fraction VR_i, we nonetheless observed values of less than 17% in all our experiments.

The remainder of the paper contains a description of the Aqueduct system and our evaluation of it. We describe related work in Section 2, and the Aqueduct system architecture in Section 3. We then present the results of our experimental evaluation in Sections 4, 5 and 6, first describing our experimental infrastructure, and then the results of testing Aqueduct with two work-

loads: one purely synthetic, and the other a real, traced e-mail application. We conclude with some observations on our findings, and some suggestions for future work, in Section 7.

2. Related work

An old approach to performing backups and data relocations is to do them at night, while the system is idle. As discussed, this does not help with many current applications such as e-business that require continuous operation and adaptation to quickly changing system/workload conditions. The approach of bringing the whole (or parts of the) system offline is often impractical due to the substantial business costs it incurs.

Perhaps surprisingly, true online migration and backup are still in their infancy. But existing logical volume managers (e.g., the HP-UX logical volume manager, LVM [22], and the Veritas Volume Manager, VxVM [27]) have long been able to provide continuing access to data while it is being migrated. This is achieved by creating a mirror of the data to be moved, with the new replica in the place where the data is to end up. The mirror is then *silvered* – the replicas made consistent by bringing the new copy up to date – after which the original copy can be disconnected and discarded. Aqueduct uses this trick, too. However, we are not aware of any existing solution that bounds the impact of migration on client applications while this is occurring in terms that relate to their performance goals. Although VxVM provides a parameter, vol_default_iodelay, that is used to throttle I/O operations for silvering, it is applied regardless of the state of the client application. High-end disk arrays (e.g., the HP Surestore Disk Array XP512) provide restricted support for online data migration [14]: the source and destination devices must be identical Logical Units (LUs) within the same array, and only global, device-level QoS guarantees such as bounds on disk utilization are supported. Some commercial video servers [16] can re-stripe data online when disks fail or are added, and provide guarantees for the specific case of highly-sequential, predictable multimedia workloads. Aqueduct does not make any assumptions about the nature of the foreground workloads, nor about the devices that comprise the storage subsystem; it provides device-independent, application-level QoS guarantees.

Existing storage management products such as the HP OpenView-Performance Manager [13] can detect the presence of performance hot spots in the storage system when things are going wrong, and notify system administrators about them – but it is still up to humans to decide how to best solve the problem. In particular, there is no automatic throttling system that might address the root cause once it has been identified.

Although Aqueduct eagerly uses excess system resources in order to minimize the length of the migration, it is in principle possible to achieve zero impact on the foreground load by applying idleness-detection techniques [9] to migrate data only when the foreground load has temporarily stopped. Douceur and Bolosky [7] developed a feedback-based mechanism called MS Manners that improves the performance of important tasks by regulating the progress of low-importance tasks. MS Manners cannot provide guarantees to important tasks because it only takes as input feedback on the performance of the low-importance tasks. In contrast, Aqueduct provides performance guarantees to applications (i.e., the "important tasks") by directly monitoring and controlling their performance.

There has been substantial work on fair scheduling techniques since their inception [23]. In principle, it would be possible to schedule migration and foreground I/Os at the volume manager level without relying on an external feedback loop. However, real-world workloads are complicated and have multiple, nontrivial properties such as sequentiality, temporal locality, self-similarity, and burstiness. How to assign relative priorities to migration and foreground I/Os under these conditions is an open problem. For example, a simple 1-out-of-n scheme may work if the foreground load consists of random I/Os, but may cause a much higher than expected interference if foreground I/Os were highly sequential. Furthermore, any non-adaptive scheme is unlikely to succeed: application behaviors vary greatly over time, and failures and capacity additions occur very frequently in real systems. Fair scheduling based on dynamic priorities has worked reasonably well for CPU cycles; but priority computations remain an *ad hoc* craft, and the mechanical properties of disks plus the presence of large caches result in strong nonlinear behaviors that invalidate all but the most sophisticated latency predictions.

Recently, control theory has been explored in several computer system projects. Li and Nahrstedt [18] utilized control theory to develop a feedback control loop to guarantee the desired network packet rate in a distributed visual tracking system. Hollot et al. [11] applied control theory to analyze a congestion control algorithm on IP routers. While these works apply control theory on computing systems, they focus on managing the network bandwidth instead the performance of end servers.

Feedback control architectures have also been developed for web servers [6][19] and e-mail servers [25]. In the area of CPU scheduling, Steere et al. [26] developed a feedback based CPU scheduler that synchronize the progress of consumers and supplier processes of buffers. In [20], scheduling algorithms based on feedback control were developed to provide deadline miss ratio guarantees to real-time applications with unpredictable workloads. Although these approaches show clear promise, they do not guarantee I/O latencies to applications, nor do they address the storage subsystem, which is the focus of Aqueduct. The feedback-based web cache manager described in [21] achieves differentiated cache hit ratio by adaptively allocating storage spaces to user classes. However, they also did not address I/O latency or data migration in storage systems.
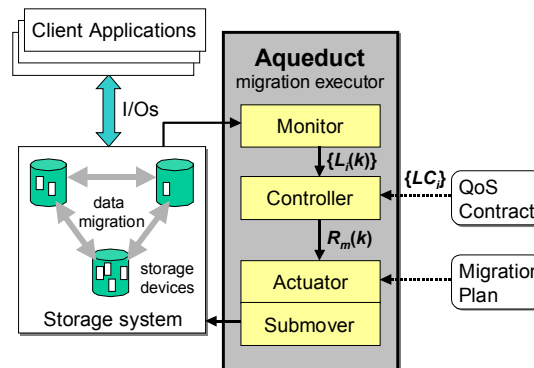
3. Aqueduct

The overall architecture of Aqueduct, and the way in which it interacts with its environment, are shown in Figure 1. It takes in a QoS contract and a migration plan, and interacts with the storage system to migrate data by using the existing HP-UX LVM's [22] primitives. As discussed above, Aqueduct relies upon the LVM silvering operation to achieve a move without having to disable client application accesses.

Ideally, Aqueduct would be integrated with the LVM, so that it could directly control the rate at which data were moved in order to achieve its QoS guarantees. Alternatively, if a dynamically-alterable parameter had been provided to control LVM's silvering rate (i.e., data movement speed), Aqueduct could have used that to effect its control. Unfortunately, neither was possible for our experiments, so we resorted to a (somewhat crude) approximation to these more tightly-coupled approaches. Fortunately, despite the overheads it imposed, it proved adequate to our goal of confirming the potential benefits of the control-feedback loop approach.

Aqueduct divides each store into small, fixed-size *substores* that are migrated one at a time, in steps called *submoves*. This allows relatively fine control over the migration speed, as substores are relatively small: we chose 32MB as a reasonable compromise between management overheads and control finesse. With LVM, we were forced to implement each substore as a complete logical volume in its own right; unfortunately, this had the undesirable property that store migrations were visible at the application level. (VxVM might have let us lift this restriction, but we did not have easy

access to a running implementation.) The resulting large numbers of logical volumes incur considerable LVM-related overheads. Nonetheless, despite the overheads, this implementation allowed us to evaluate the key part of the Aqueduct architecture – the feedback control loop – which was the primary point of this exercise.

Figure 1 The internal structure of the Aqueduct migration executor, and its relationship to the external world.

3.1. Feedback control loop in Aqueduct

The Aqueduct *monitor* component is responsible for collecting the sampled latency of each store at the end of each sampling period, and feeding these results to the *controller*. We were able to extract this data directly from an output file periodically generated by our workload generation tool; but it could also have been obtained from other existing performance monitoring tools (e.g., the GlancePlus tool of HP OpenView [13]).

The *controller* compares the sampled latencies for the time window $((k-1)W, kW)$ with the QoS contract, and computes the submove rate $R_m(k)$ (the *control input*) to be used during the next sampling period $(kW, (k+1)W)$. Intuitively, Aqueduct should slow down data migration when some sampled store latencies are larger than their corresponding contracts, and speed up data migration when latencies are smaller than the corresponding contracts for all stores. The controller computes the submove rate based on the sampled store latencies so that the sampled store latencies stay close to their corresponding contracts. Aqueduct incorporates an *integral controller*, a well-studied law in control theory [8]; integral controllers are typically robust in the presence of a wide range of workload variations. It operates as follows:

1) For each store i ($0 \leq i < N$) in the system, compute its error
$$E_i(k) = P * LC_i - L_i(k),$$

where P ($0 < P < 1$) is a configurable parameter, and $P*LC_i$ is called the *reference* in control theory. More negative values of $E_i(k)$ represent larger latency violations.

2) Find the smallest (i.e., potentially most negative) error $E_{min}(k)$ among all stores:
$$E_{min}(k) = \min\{E_i(k)| \ 0 \le i < N\};$$
thus taking account of the worst contract violation observed.

3) Compute the submove-rate according to the integral control function (K is another configurable parameter of the controller):
$$R_m(k) = R_m(k\text{-}1) + K*E_{min}(k);$$

4) Notify the actuator of the new submove rate $R_m(k)$.

Because the control input $R_m(k)$ is computed from the $E_i(k)$ corresponding to the worst violation, it forces the system to satisfy its latency goals by arranging for E_{min} to converge to zero. Thanks to random workload variations, store I/O latencies will typically oscillate around the reference value, so instead of choosing the actual latency target LC_i as the reference, the controller uses a slightly smaller target: $P*LC_i$. The value of P is related to the burstiness of the workload: the more bursty a workload is, the smaller P should be, to give the controller enough leeway to avoid contract violations. On the other hand, overly small values of P will result in an overly conservative controller, and therefore slow down migration. In our experiments, we observed that a P between 0.8 and 0.9 was sufficient to achieve satisfactory violation fractions for significantly different workloads**.**

Parameter K needs to be tuned to achieve *stability* (i.e., to prevent the submove rate and sampled latencies from oscillating excessively) and *short settling time* (i.e., fast convergence of the output to the reference). This can be done using systematic, standard control theory techniques. An example is provided in Section 5.1. A similar tuning method was described in detail, and applied to a real-time CPU scheduler in [20]. Aqueduct could be extended in a fairly straightforward way to set (and adjust) K automatically, using an online estimation of the gain [5] in order to handle different categories of workloads without the need for precomputed parameter values.

The last module in Figure 1 is the *actuator*. It executes a migration plan at the submove rate computed by the controller. During the sampling period (kW, $(k+1)W$), the actuator enforces the submove rate $R_m(k)$ by sleeping for ($W/R_m(k)$ - T_j) time units between the end of submove j and the start of the next, where T_j is the time it took to complete submove j.

4. Experiment overview

We evaluated the performance of Aqueduct in our storage area network, using both a synthetic workload and an I/O trace from a production e-mail server.

The hardware used for our tests includes an HP FC-60 disk array [15] with 512 MB of cache, two redundant controllers, and six disk enclosures with 5 disks each, for a total unprotected capacity of 1.05TB. All LUs in the array were 6-disk RAID-5s with 16-KB stripe units. The FC-60 array was connected to a Brocade Silkworm 2800 switch via two 1Gb/s Fibre Channel links. Both Aqueduct and the load generators ran on the same HP 9000-N4000 server, which has eight 440 MHz PA-RISC 8500 processors and 16GB of RAM. The host ran the HP-UX 11.0 operating system. We used our own workload-generation tool, Buttress, which is capable of generating synthetic workloads and replaying an existing I/O trace with very high fidelity.

We compared Aqueduct against two baselines:

- **Whole-store**, moves a whole store in each step as fast as possible, with no delays between store-moves; stores are not divided into smaller sub-stores. This is designed to reflect what a system administrator would do when migrating data by hand or by running simple scripts.

- **Sub-store** is similar to Whole-store, but divides each store into substores and performs each move as a sequence of submoves. It is a fairer baseline for comparison with Aqueduct because it uses the same number of logical volumes (substores) and hence incurs similar amounts of LVM overhead.

In these experiments, all stores were given the same latency contract, LC = 10 ms, and we always used a sampling period (W), of 60 seconds, and a substore size of 32MB. The following table lists the configurable parameters we used for the two workloads:

	Synthetic	OpenMail
K	1.09	0.36
P	0.90	0.80

Since the OpenMail workload is more sensitive to changes in the submove rate than the synthetic workload, we tuned K to be smaller for the OpenMail workload based on control theory (described in Section 5.1). We found that the first value we tried for P (0.9) was adequate for the synthetic workload, but not for OpenMail—the second trial for OpenMail resulted in the final, slightly smaller P = 0.8. This was not unexpected, as OpenMail is more bursty than our deliberately well-behaved synthetic workload.

We define the *victim latency VL(k)* as the highest sampled latency of all stores in the kth sampling period, i.e., $VL(k) = \max\{L_i(k): 0 \leq i < N\}$. In this special case in which all stores share a same latency contract *LC*, all stores satisfy their contracts if and only if the victim latency stays lower than the contract. Similarly, the *average victim latency AVL* is the average of the values of *VL(k)* over all *M* sampling periods during the migration. *AVL* reflects the "correctness" of the migration speed. Ideally, *AVL* should be close to the latency contract. If *AVL* > *LC*, the migration runs too fast and causes excessive contract violations. On the other hand, if *AVL* < *LC*, the migration could have run faster without violating the latency contract.

5. Synthetic workload experiments

Figure 2 illustrates the initial state and the migrations that were effected in this test. The synthetic workload is composed of multiple streams with fixed, identical parameters, and tests Aqueduct in the presence of deliberately steady workloads. We configured two LVM volume groups, aq0 and aq1. All stores are 640 MB in size, and are therefore divided into 20 substores each. Group aq0 contains six stores. In the initial assignment, three of them (the *migrate-stores* M0, M1, M2) are mapped onto logical unit LU1 of the disk array; the remaining three stores of aq0 (the *fixed-stores* F0, F1, F2) are in another logical unit LU0.

These experiments emulate the following use scenario: assume that we find that LU1 is likely to fail (e.g., by using system monitoring tools such as [17]), so we want to move the migrate-stores in LU1 to a new logical unit LU3. Hence, the migrate- and the fixed-stores belong to the same LVM volume group. We hypothesized that stores contained within the same volume group where data is being migrated would suffer some performance penalty from LVM overhead, even if they were not being migrated. To test that assumption, we created group aq1 on LU2, whose stores (the *alone-stores* A0, A1, A2) should not be affected because they are totally separate from aq0.

To generate the workload from the client applications, we simulated the file system workload described in [2] by issuing two synthetic streams on each store. Each stream has a Poisson arrival process, 16KB request size, with a run count of 3 (this is the average number of consecutive I/O requests performed on consecutive addresses), 64% of the requests are reads, and the request rate is 32/second.
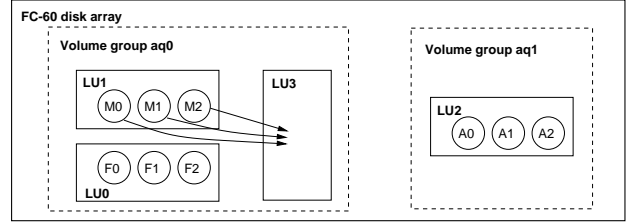
Figure 2 Placement of stores on disk array logical units for the synthetic workload, and the migration that was performed.

5.1. Tuning *K*

We demonstrate here how the parameter *K* was selected for the synthetic workload using the Root-Locus design technique [8]. The *controlled system* includes the storage system, the monitor, and the actuator. The output is the victim latency *VL(k+1)*, i.e., the sampled latency of the store with the smallest error $E_{min}(k+1)$ in the sampling period $(kW, (k+1)W)$. The input to the controlled system is the submove rate $R_m(k)$ in the sampling period $(kW, (k+1)W)$. ($R_m(k)$ instead of $R_m(k+1)$ is used to denote the submove rate in $(kW, (k+1)W)$ because the controller outputs $R_m(k)$ at time kW instead of $(k+1)W$. In our design, we approximate the controlled system with the following linear model:

$$VL(k+1) - VL(k) = G(R_m(k) - R_m(k-1)) \tag{1}$$

The *process gain*, *G*, is the derivative of the output *VL(k+1)* with respect to the input $R_m(k)$. *G* represents the sensitivity of the victim latency with regard to the change in submove rate. We approximate *G* by running a set of system profiling experiments. In each run, migration is performed at a fixed submove rate, and different submove rates are used in different runs. The average victim latencies observed throughout migration for different submove rates are plotted in Figure 3. Using linear regression, we estimate that *G* = 1.12 with an R^2 of 99% for the synthetic workload.

We now transform the controlled system model into the *z*-domain, which is amenable to control analysis. The controlled system model in Equation 1 is equivalent to the following transfer function from $R_m(z)$ to *VL(z)* in *z*-domain: $H(z) = G * z^{-1}$. The integral controller is transformed to the following transfer function from the minimum error $E_{min}(z)$ to the submove rate, $R_m(z)$, in the *z*-domain: $C(z) = Kz/(z-1)$.

It follows that the whole feedback control system composed of the controlled system and the integral controller is modeled as the following transfer function from the reference to the victim latency:

$$H_C(z) = \frac{C(z)H(z)}{1 + C(z)H(z)} \qquad (2)$$

Assume that all the stores share a common contract $P*LC$, the z-transform of the victim latency is:

$$VL(z) = H_C(z) * P * LC * \frac{z}{z-1} \qquad (3)$$
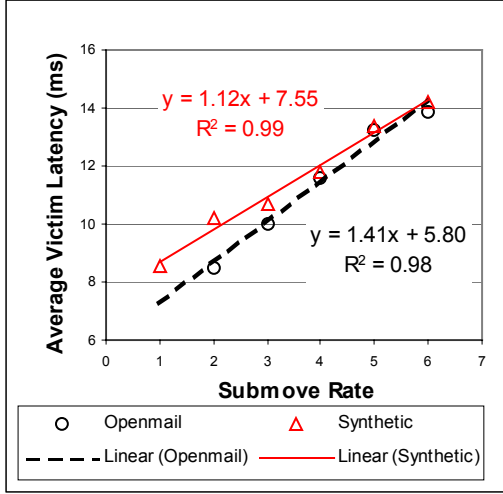
Figure 3 Victim latency as a function of submove rate for the synthetic workload and openmail workload.

Given the dynamic model of the closed loop system, we tune the control parameter K analytically using linear control theory [8], which states that the performance of a system depends on the poles of its closed loop transfer function. Since the closed loop transfer function (Equation 2) of Aqueduct has a single pole $p = 1-KG$, we can set p to the desired value by choosing the right value of K. The sufficient and necessary condition for Aqueduct to guarantee stability is: $|p| < 1 \Leftrightarrow 0 < K < 2/G$. The settling time represents the time it takes to converge the victim latency to the reference. A smaller settling time leads to a faster response to workload variations. The settling time is determined by the damping ratio of the closed loop system. A larger G (e.g., in a workload whose latency is more sensitive to the submove rate) needs a smaller K to get the same pole and achieve the same level of stability and settling time. Using the root-locus method, we set $p = -0.22$ by choosing $K = (1-p)/G = 1.09$ to guarantee stability and a short settling time. The OpenMail has a larger gain than the synthetic workload, so it benefits from a lower value of K.

5.2. Experimental results

We now explore the results of applying Aqueduct to migrating data that is being accessed by the synthetic workload.

The sampled latencies of store M0 in typical runs of Aqueduct and the baselines are illustrated in 0. (We pick M0 because it is the store most affected by migration, as shown in Figure 6.) Whole-store causes long latencies throughout migration; latencies are especially severe near the end of the migration when they jump to 25.17 ms. This is because near the end of the migration, more application I/Os target at the new logical unit, LU3, and contend more severely with Whole-store which *writes* into LU3 in parallel. Interestingly, in the beginning of migration, more application I/Os target at the replaced logical unit LU1 and contend with Whole-store which *reads* data from LU1, but the impact of migration on latencies is less severe. This is because writes are more expensive than reads on RAID5 (especially if they are small, as done by LVM when silvering), and therefore migration consumes more resources on LU3 than on LU1. Although Whole-store completes data migration within the shortest time, it violates the latency contract (10 ms) throughout the data migration period.
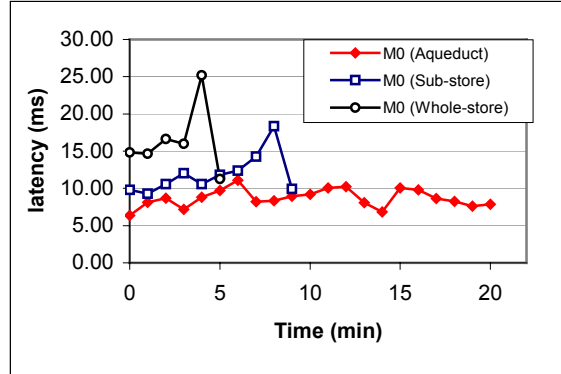
Figure 4 Data from three typical runs in the synthetic workload experiments, showing foreground application I/O latency on store M0 during the execution of the Aqueduct, Sub-store, and Whole-store migration algorithms.

Sub-store migrates data more slowly, and with smaller impact on client applications, than Whole-store. However, the latency contracts are still violated in most sampling periods. Since neither Sub-store nor Whole-store sleeps between subsequent (sub)moves, we attribute the difference between their migration times and interference on client applications to the overhead of managing large number of logical volumes in Sub-store—which is slowed down by this effect.

In comparison, in the case of Aqueduct, the latency of M0 stays below the latency contract in most of the sampling periods. This result demonstrates that Aqueduct effectively reduces migration's impact on client applications. Note that the latency of M0 stays close to

the contract latency. This indicates that, although Aqueduct has a longer migration time than the baselines, it achieves a submove rate that is close to the maximum allowed by the QoS contract.

To demonstrate the quality of control by Aqueduct, we plot the traces of sampled latency on M0 and submove rate (the control input) during the same sample run. We can see that Aqueduct effectively keeps latency close to the reference (9 ms) by dynamically adapting the submove rate—peaks and valleys are strongly correlated in the two curves. Furthermore, Aqueduct achieves satisfactory stability because it does not cause excessive oscillation in submove rate or latency throughout the run.
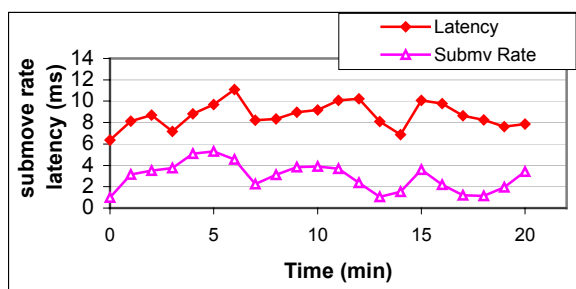
Figure 5 Foreground application latency on store M0 during a sample Aqueduct migration, together with a plot of the rate at which sub-stores were being moved (in moves per minute).

5.3. QoS guarantees

We now evaluate how Aqueduct provides QoS guarantees for the synthetic workload. Every data point presented in this section and Section 5.4 is the mean of 5 repeated runs. We also report the 90% confidence intervals for every data point.
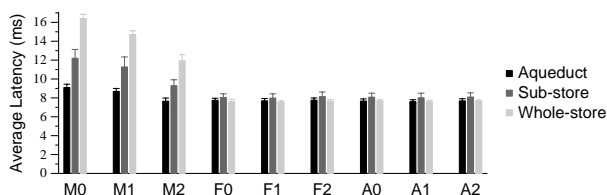
Figure 6 Average application I/O latencies in the synthetic workload experiments.

The average latencies for Aqueduct and the baselines are illustrated in Figure 6. The latencies of the alone-stores and the fixed-stores are similar, and therefore the impacts of LVM overhead on fixed-stores are negligible. Migration has negligible impacts on the average latencies of the fixed-stores or the alone-stores with all migration methods. However, different migration methods perform differently on the migrated stores. In

particular, Whole-store achieves an average latency on M0 of 16.4 ($\pm$0.5) ms,, which is 80% higher than Aqueduct's 9.1 ($\pm$0.4) ms. Similarly, Sub-store achieves an average latency of 12.2 ($\pm$0.9) ms, or 34% higher than Aqueduct. More importantly, Aqueduct's average latencies of all stores are lower than the latency contract of 10 ms, while the average latencies of Sub-store and Whole-store are higher than the contract in two and three migrate-stores, respectively.
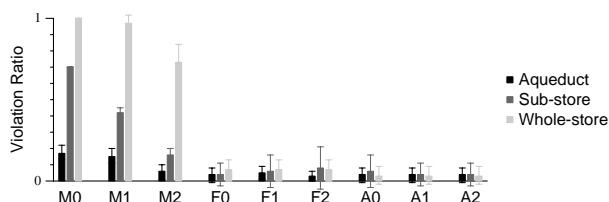

Figure 7 Contract violation fractions in the synthetic workload experiments.

The contract violation fractions for the synthetic workload are shown in Figure 7. Whole-store violates the latency contract in all sampling periods during migration. While Sub-store achieves a lower contract violation fraction due to LVM overheads in data migration, it still causes a much higher violation fraction than Aqueduct. In particular, Sub-store violates the latency contract in 70% ($\pm$0%) of all sampling periods during migration, while Aqueduct only violates 17% ($\pm$5%) of all sampling periods. The contract violation fraction is important because a lower value means that client applications suffer violations less frequently and hence the storage service has more acceptable performance.

5.4. Migration efficiency

As expected, Aqueduct provides QoS guarantee to applications at the expense of slowing down data migration. Figure 8a shows that it takes Aqueduct 1219 ($\pm$43) sec on average to complete the migration plan, while Sub-store only needs 556 ($\pm$3) sec. Sub-store migrates data more slowly than Whole-store due to the LVM overhead.
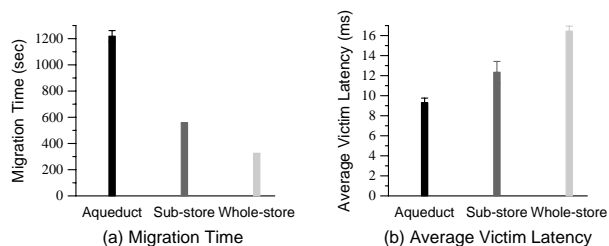
Figure 8 Migration times in synthetic workload experiments.

In order to determine whether Aqueduct achieves the maximum speed allowed by the QoS contracts, we look at average victim latencies. The rationale is that, to guarantee that no stores violate the latency contract, the victim latency must be the same or lower than the latency contract, i.e., the latency contract must be an upper bound on the victim latency. Figure 8b shows that Aqueduct achieves an average victim latency of 9.30 (± 0.46) ms, which is only 7% lower than the latency contract. Given that the average submove rate is below 3 submoves/min., even an increase of 1 submove/min. in the control input would result in service contract violations. This result shows that Aqueduct's bound is tight: Aqueduct is not overly conservative, and it achieves a migration speed close to the maximum that is possible given the constraint of providing latency guarantees. In addition, note that the average victim latency is close to the actual reference to the controller ($P*LC = 9$ ms), which shows that the controller correctly enforces the reference.

In summary, the synthetic-workload experiments demonstrate that Aqueduct can effectively provide latency guarantees to applications having steady, regular access patterns, while performing online data migration efficiently. Aqueduct guarantees the average latencies of all stores to be lower than the latency contract, and achieves a contract violation fraction of no more than 17%. For the same migration plan, Whole-store and Sub-store cause average latencies higher than the latency contract in migrated-stores and contract violations as high as 100% and 70%, respectively. In term of migration efficiency, Aqueduct achieves a migration speed close to the maximum allowed by the latency contract.

6. OpenMail experiments

The OpenMail workload was originally gathered by tracing an e-mail server running HP OpenMail [12]. The original workload trace was collected on an HP 9000 K580 server system with an I/O subsystem comprised of four EMC Symmetrix 3700 disk arrays. The server was sized to support a maximum of about 4500 users, although only about 1400 users were actively accessing their email during the trace collection period, which corresponded to the server's busiest hour of the day. The majority of accesses in the trace are to the 640 GB message store, which is striped uniformly across all of the arrays.

In order to create a trace comparable to our synthetically generated workloads, we replayed the portion of the original trace corresponding to a single representa-

tive array on our FC-60 array. Since the LVM on HP-UX 11.0 has a limitation that each volume group can contain at most 255 logical volumes, and each logical volume corresponds to one substore (32 MB each) in our current Aqueduct prototype, we shrank the sizes of the corresponding stores proportionally to a total size of 3.8 GB to fit them into one volume group. (This size limitation can be fixed by a future Aqueduct implementation with modifications on the LVM.)

This workload has significantly more complex behaviors than our synthetic one. The OpenMail system being traced kept a small amount of metadata (an index table) at the beginning of the message store's address space, and filled up the remainder with e-mail messages. For each email retrieval request from a user, or on each incoming email, the server accesses the initial index table and then jumps to actually access the message, to a random location uniformly distributed across the upper portion of the store. Consequently, the small amount of metadata becomes a hotspot that gets accessed much more frequently than the other data.
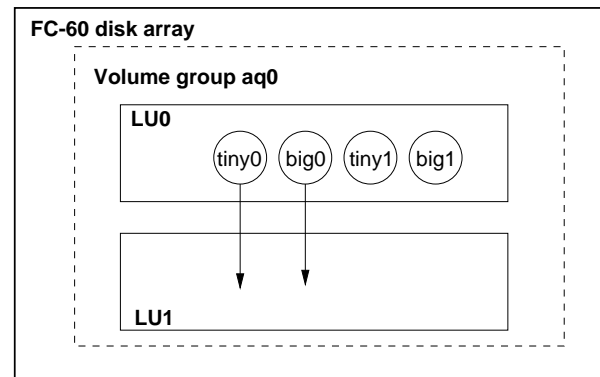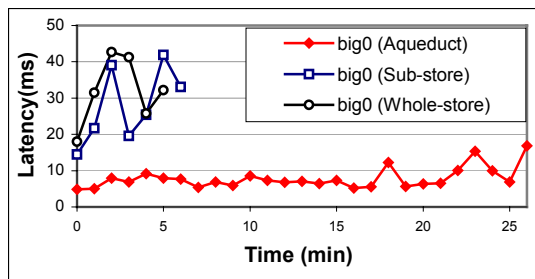

Figure 9 Store migration plan for the OpenMail workload.

We create one volume group, aq0, which includes 4 stores called tiny0, tiny1, big0, and big1, respectively. tiny0 and tiny1 are 96 MB each, and big0 and big1 are 1854 MB each. In the initial assignment, all the stores are located on a single logical unit LU0.

The OpenMail experiments emulate a LU-addition scenario. We model the case of wanting to increase the server capacity by adding a new Logical Unit, LU1, to the array. To make use of the new LU, we migrate two stores, tiny0 and big0, from LU0 to LU1.

Similarly to the synthetic workload, we approximate the process gain, G, for the openmail workload with a set of system profiling experiments. In each run, migration is performed at a fixed submove rate, and different submove rates are used in different runs. The average victim latencies observed throughout migration

for different submove rates are plotted in Figure 3. Using linear regression, we estimate that $G = 1.41$ with an R^2 of 98% for the openmail workload. Compared with the synthetoc workload, the process gain of the openmail workload larger. This result means that openmail is more senstive to the impacts of migration and therefor a smaller K is needed. In our experiments we set $K = 0.36$ (corresponding to a pole $p = 0.49$) to guarantee stability and a short settling time for the openmail workload.


Figure 10 Sampled latency for the OpenMail workload during migration.
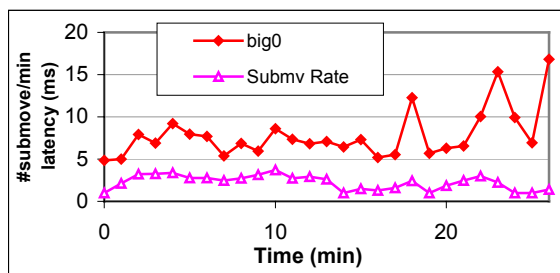
Figure 11 Sampled latency and substore move rate (in moves per minute) of Aqueduct for a typical migration of the OpenMail workload.

The sampled latencies of store big0 in typical runs of Aqueduct and the baselines are illustrated in Figure 10. Both Whole-store and Sub-Store cause extremely long latencies on big0 and violate the latency contract throughout migration. In comparison, with Aqueduct, big0's latencies stay below the latency contract (10 ms) in most sampling periods. Figure 11 shows the traces of sampled latency on big0 and submove rate during the same sample run. Aqueduct effectively keeps latency close to the reference (8 ms) by dynamically adapting the submove rate without causing excessive oscillation.

In the following subsections, we present the detailed evaluation results of Aqueduct in the OpenMail experiments. Every data point presented in this section is the mean of five repeated runs. The 90% confidence intervals are also plotted.
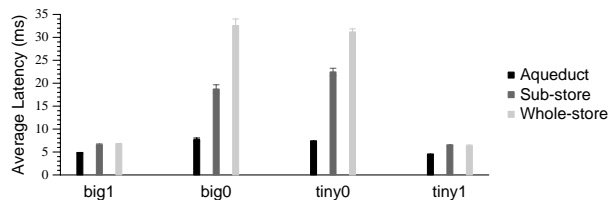
Figure 12 Average application I/O latencies during migrations for the OpenMail workload.

6.1. QoS guarantees

The average latencies in the OpenMail experiments are shown in Figure 12. The OpenMail application is much more sensitive to data migration overheads than the synthetic workload. For example, Sub-store increases the average latencies of accesses to the migrated-stores, big0 and tiny0, to 18.74 ($\pm$0.92) ms and 22.46 ($\pm$0.81) ms – which are 87% and 125% higher than the latency contract (10 ms), respectively. In comparison, Aqueduct achieves an average latency no higher than 7.70 ($\pm$0.36) ms, or 23% lower than the contract in all stores. This result demonstrates the efficacy of Aqueduct in applications that are very vulnerable to online data migrations.
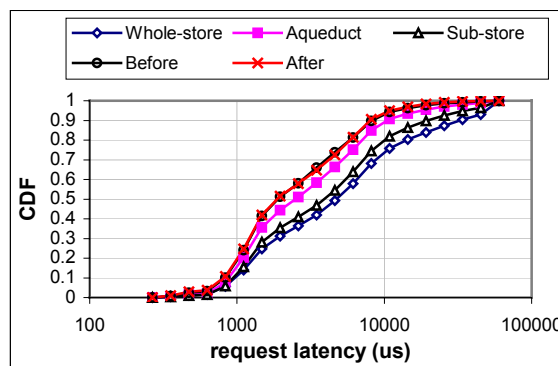

Figure 13 Cumulative distribution of I/O times during data migration by the three different schemes for the OpenMail workload, across the entire workload. Aqueduct has the smallest impact on the I/O request latencies. The "before" and "after" values on this plot are for the sub-store case, but the differences with the other alternatives are almost too small to show. Note the log scale on the x-axis.

A study of the distribution of I/O request latencies during a migration (Figure 13) shows that the effect of Aqueduct is to reduce the number of requests that suffer significantly longer I/O times: application I/Os queued behind a data migration operation result in large delays.

The contract violation fractions for the different migration algorithms are shown in Figure 14. Aqueduct sig-

nificantly reduces the contract violation fractions of the migrated stores, big0 and tiny0. For example, the contract violation fraction of tiny0 is reduced from 98% with Sub-store to only 7% with Aqueduct. Thus, Sub-store causes applications to suffer contract violations in almost every sampling period during data migration, while contract violations rarely occur with Aqueduct.
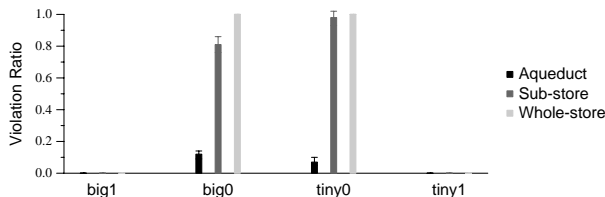

Figure 14 Contract violation fractions in OpenMail experiments.

6.2. Migration efficiency

As shown in Figure 15a, Aqueduct increases the migration time more significantly in the case of OpenMail than in the case of the synthetic workload. Because OpenMail is affected more severely by migration, Aqueduct is forced to perform migration more slowly.

The average victim latency (see Figure 15b) of Aqueduct is 8.46 (±0.31) ms, or 15% lower than the latency contract. Again, the migration speed is close to the maximum speed allowed by the latency contract. We also note that the average victim latency is within 6% of the reference (8 ms), which shows that the Aqueduct controller is able to successfully track the control reference even in the presence of bursty workloads such as OpenMail.

In summary, the OpenMail experiments demonstrate that Aqueduct provides latency guarantee to real-world applications that are especially sensitive to migration. In particular, Aqueduct meets its QoS guarantees, and achieves an average victim latency that is only 15% below the latency contract. As in the synthetic experiments, Aqueduct performs migration at a speed close to the maximum allowed by the latency contract.
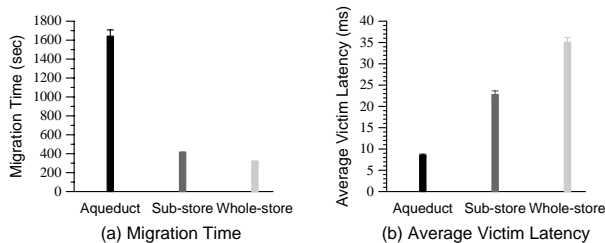
Figure 15 Migration time in OpenMail experiments.

7. Conclusions and future work

We have developed Aqueduct, an online data migration architecture that provides QoS guarantees to client applications. Aqueduct features a feedback control loop that dynamically adapts migration speed to maintain performance guarantees in the presence of workload and system variations.

We evaluated a prototype on a real storage system, using a high-end host and disk arrays similar to the ones used in large enterprise installations. Our experiments show that Aqueduct successfully provides QoS guarantees in term of bounded average latencies, while causing only a small percentage of contract violations. Aqueduct reduces the average I/O latency experienced by client applications by as much as 76% with respect to the traditional method: while accesses to a store in an e-mail server have an average I/O latency of 32.6 ms while a non-adaptive migration is in progress, accesses to the same store have an average latency of only 7.7 ms with Aqueduct. Aqueduct also reduces the violation fraction from 100% to only 12%. Furthermore, Aqueduct performs data migration very close to the maximum speed allowed by the latency contract, as evidenced by the small slack of only 15% between the average victim latency and the latency contract.

Potential future work items include a more general implementation that interacts with performance monitoring tools, developing a low overhead mechanism for finer-grain control of the migration speed, making the controller self-tuning to handle different categories of workloads, and implementing a new control loop that can simultaneously bound latencies and violation fractions.

References

[1] 3Com Corporation, "Gigabit Ethernet Comes of Age," Technology white paper, June 1996.

[2] G. A. Alvarez, E. Borowsky, S. Go, T. H. 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*, vol. 19, no. 4, November 2001.

[3] E. Anderson, J. Hall, J. Hartline, M. Hobbs, A. Karlin, J. Saia, R. Swaminathan, and J. Wilkes. "Experimental Study of Data Migration Algorithms," *5th Workshop on Experimental Algorithms*, August 2001.

[4] ANSI, "Fibre Channel Arbitrated Loop," Standard X3.272-1996, April 1996.

[5] T. F. Abdelzaher, "An Automated Profiling Subsystem for QoS-Aware Services," *IEEE Real-Time Technology and Applications Symposium*, pp. 208-217, June 2000.

[6] T. F. Abdelzaher and N. Bhatti, "Web Server QoS Management by Adaptive Content Delivery," *International Workshop on Quality of Service*, pp. 216-225, 1999.

[7] J. R. Douceur and W. J. Bolosky. "Progress-based regulation of low-importance processes," 17th *ACM Symposium on Operating Systems Principles*, pp. 247-260. Dec 1999.

[8] G. F. Franklin, J. D. Powell and M. L. Workman, *Digital Control of Dynamic Systems (3rd Ed.),* Addison-Wesley, 1998.

[9] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and John Wilkes. "Idleness is not Sloth," *Winter'95 USENIX Conference*, pp 201-212, Jan. 1995.

[10] S. Gribble, G. Manku, E. Roselli and E. Brewer, "Self-similarity in File Systems," *SIGMETRICS'98*, pp. 141-150, April 1998.

[11] C. V. Hollot, V. Misra, D. Towsley, and W. Gong, "A Control Theoretic Analysis of RED," *IEEE INFOCOM*, pp. 1510-1519, April 2001.

[12] Hewlett-Packard Company, *Openmail*, http://www.OpenMail.com/cyc/om/00/index.html.

[13] Hewlett-Packard Company, *HP OpenView Homepage*, http://www.openview.hp.com/.

[14] Hewlett-Packard Company, *HP SureStore E Auto LUN XP User's Guide*, May 2000.

[15] Hewlett-Packard Company, *HP SureStore E Disk Array FC60 – Advanced User's Guide*, Dec 2000.

[16] International Business Machines Corp., Content Manager VideoCharger Administrator's Guide, January 2001.

[17] International Business Machines Corp., *Drive Fitness Test*, August 1999.

[18] B. Li and K. Nahrstedt, "A Control-based Middleware Framework for Quality of Service Adaptations," *IEEE Journal of Selected Areas in Communication, Special Issue on Service Enabling Platforms*, 17(9), pp. 1632-1650, Sept. 1999.

[19] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, "A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers," *IEEE Real-Time Technology and Applications Symposium*, pp. 51-62, June 2001.

[20] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Journal of Real-Time Systems, Special Issue on Control-Theoretical Approaches to Real-Time Computing*, 22(3), March 2002.

[21] Y. Lu, A. Saxena, and T. F. Abdelzaher, "Differentiated Caching Services; A Control-Theoretical Approach," *International Conference on Distributed Computing Systems*, pp. 615-622, April 2001.

[22] T. Madell, *Disk and File Management Tasks in HP-UX*, Prentice-Hall, 1997.

[23] J. Nagle. "On packet switches with infinite storage," *IEEE Trans. on Communications*, vol. 35, no. 4, pp. 435-38, April 1987.

[24] G. Papadopoulos, "Moore's Law Ain't Good Enough," keynote speech at *Hot Chips X*, August 1998.

[25] S. Parekh, N. Gandhi, J. L. Hellerstein, D. Tilbury, T. S. Jayram, J. Bigus, "Using Control Theory to Achieve Service Level Objectives in Performance Management," *IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[26] D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, J. Walpole, "A Feedback-driven Proportion Allocator for Real-Rate Scheduling," *Symposium on Operating Systems Design and Implementation,* pp. 145-158, Feb 1999.

[27] Veritas Software Corp., *Veritas Volume Manager*, http://www.veritas.com/us/products/volumemanager.