Automated Storage Management with QoS Guarantee in Large-scale Virtualized Storage Systems*

Lin Qiao * Balakrishna R. Iyer § Divyakant Agrawal † Amr El Abbadi †

§ IBM Silicon Valley Lab
San Jose, CA
balaiyer@us.ibm.com

† Department of Computer Science
University of California, Santa Barbara
{agrawal, amr}@cs.ucsb.edu

* IBM Almaden Research Center
San Jose, CA
lsqiao@us.ibm.com

Abstract

Storage virtualization in modern storage systems allows variability in the number of "physical" disks supporting a single "virtual" disk. In practice, IO workloads vary with time. Presuming the evolution of reasonable predictive models with the power of accurately predicting IO workload, it may be argued that it is straightforward to compute the number of disks needed at any time to satisfy QoS constraints. However, because disks contain data, the data also needs to be reallocated amongst disks. Not only does data migration need to be scheduled ahead but it must also be scheduled in such a way that QoS violations do not occur because of the extra migration IOs. In this paper, we present a novel analytic framework, PULSTORE*, for autonomically managing the storage to balance both cost and performance. Given the workload characteristics of an application and storage QoS requirements, our* PULSTORE *framework yields an optimal schedule for adding and removing disks to support a time varying IO workload without QoS violations. A case study based on real IO traces shows that* PULSTORE *is very effective in achieving both QoS and utility goals.*

1 Introduction

Traditionally, storage has been purchased and attached to a single computer system. Such storage is accessible only through the computer system to which it is locally attached. In the last 10 years, especially in corporate data centers, storage is being increasingly purchased independent of the processors, and independently managed and administered. Because of the standardization of disk IO protocols, storage can be easily shared amongst various heterogeneous processors running different applications. The shared storage is accessed over a network interconnecting the processors to the shared disk subsystem, known as the *storage area network* - a network on which processors send IO calls to virtual disks. It is the task of the storage controller to manage the mapping of virtual disks to physical disks, a task known as *storage virtualization*, similar to memory virtualization of processors. The storage virtualization layer has been exploited to provide diverse storage functions.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*The work was supported in part by NSF awards IIS-02-23022, IIS-02-20152 and EIA-00-80134. Lin Qiao was also supported in part by IBM Corporative Fellowship.

The storage virtualization layer may be used not only statically but also dynamically to change the mapping of virtual disks to physical disks. This can be exploited to dynamically change the physical disk configuration. Many IO workloads exhibit cyclic behaviors, and alternate between bursts of high activity and periods of low activity. A significant number of practical instances show that the IO workload lends itself to be predictable. For example, the IO workload for a week extracted from HP Cello92 IO traces [11] collected from an HP-UX file server exhibits highly repetitive behaviour. It has also been observed that there is an order of magnitude difference between the peak and average IO rates. Configuring the IO sub-system for the peak leads to over-provisioning and waste. In a virtualized storage management system, it is possible to provision on demand, especially when the storage is shared amongst multiple applications, each of which would have peak requirements at different times.

If a reasonable prediction of IO workload can be made, the storage virtualization layer could optimize the mapping of physical disks to virtual disks to satisfy applications' IO response time requirements. This problem is usually referred to as *on-demand utility provisioning* [10]. This is analogous to the problem of processor allocation to shared concurrent workloads. The difference is that disks contain data. Hence if we change the number of disks supporting a specific collection of data we also need to redistribute the data. This task involves data movement, which generates even more IO. Under some circumstances, it may not always be advisable to change the number of disks. If the number of disks are to be changed, then the change needs to be done in advance. When must it be done? These are questions that will be answered in this paper.

In this paper, we tackle the problems of moving data in a storage hierarchy under both capacity/performance constraints and on-demand resource provisioning constraints. The challenges are two folds. First, for a single data movement action, as it interacts with the applications, we need to limit its impact on application performance. In particular, we need to control the invocation time of a data movement such that there is no performance constraint violation during and after the data movement. Second, as workload varies over time, it is likely that the previous storage provisioning action may have been either too small or too large, resulting in either capacity/performance constraint violation (under-provisioning) or wasteful storage configuration (over-provisioning). Hence it is important to dynamically generate a sequence of carefully tuned data movement actions in order to adapt to the changing workload. To solve these problems, we have developed an analytic framework, namely PULSTORE, to maintain QoS constraints, while avoiding waste of resources.

2 The System Overview

In this paper, we specifically address the problem of balancing the conflicting goals of the storage utility cost and QoSS requirements. Our study is based on a hierarchical storage structure, storage QoS goals, an online data migration model, and a storage utility cost model.

2.1 Storage Architecture

Modern applications access the storage in terms of logical disks, which are then transparently mapped to the physical disks by the storage virtualization engine [3]. Figure 1 depicts the architecture of such a hierarchical storage system. Here the virtualization engine can reside in a storage controller or the storage management layer of a DBMS.

Each layer in the hierarchy contains a pool of identical storage devices, called a *storage pool*. Moving upward through the hierarchy, each layer provides faster access speed but is more expensive. As we move downward through the hierarchy, the cost decreases and so does the access speed. Such a storage hierarchy scheme provides flexible control over the tradeoffs between access speed and cost through data movements. The overall objective of the storage hierarchy is to provide the fastest average access speed with the least expensive average cost of data. Such a storage model is widely used in large-scale data centers [2].

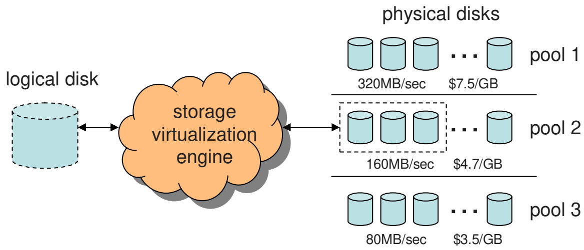


Figure 1: Virtualization for Hierarchical Storage Pools

2.2 Workload and QoSS

An IO workload is a time series of IO requests. It is characterized by the IO request size, IO arrival rate, etc.The storage level quality-of-service (QoSS) specifies the IO performance for a particular storage object that must be guaranteed under any workload. QoSS goals are associated with logical disks [5]. We use L_{QoS} to denote the QoSS goal on IO latency for a logical disk. To enforce the QoSS goals, it is crucial to predict the performance outcome for a given workload and storage system configuration. There are numerous models [4, 7, 1] on models that predict performance outcome. In these models, the IO latency can be represented as a function of the workload and the logical disk configuration. Since these two parameters are a function of time, latency can simply be predicted as a function of time, denoted as $L(t)$. The QoSS goal is expressed as a bounded latency: i.e., $L(t) \leq L_{QoS}$ at any time t.

2.3 Online Data Migration

Data migration is initiated when the storage system experiences performance degradation or anticipates disk failure. Furthermore, data migration needs to be *online*, i.e., no application should be interrupted during data migration. In our model, one logical disk is striped across all the physical devices assigned to it. We study two aspects which directly impact the performance of a striping storage system. We can either increase the physical disk speed or the striping width to decrease the IO latency, or vice versa to increase the IO latency[1]. The change of physical disk speed is realized through the data movement across pools (inter-pool migration), and the change of the striping width is implemented through the data movement inside a pool (intra-pool migration) by adding or removing disk(s) from the current configuration. A migration that reduces the application latency is called *up migration*. A migration that reduces resource utility cost is called *down migration*.

We now describe a typical online data migration scheme, which has been exploited in commercial DBMSs, such as IBM DB2 UDB [6]. The data location is maintained by a data element map. Once a data element is moved from one source disk to another destination disk, the data element map is updated. Therefore, part of the workload is directed to the destination disks and the other part of the workload continues to be directed to the source disks until migration completes.

2.4 Storage Utility Cost

The storage cost over time for a particular logical disk is called the *resource utility cost*, or $\mathcal{U}$. It is the summation of the cost of the provisioned storage system, C_t, at each time t over the whole time period.

$$\mathcal{U} = \sum_t C_t$$

Our goal is to minimize $\mathcal{U}$ while guaranteeing that no QoSS violation occurs.

[1]Changing the striping unit size dynamically for varying workloads has been studied [12]. We do not address this problem here.

3 PULSTORE

We present the design of a system, named PULSTORE [8, 9], whose goal is to produce a migration sequence so that the performance outcome satisfies the QoSS specification while minimizing the resource utility cost.

3.1 Requirements

In order to adapt to the variations in workload while providing QoSS guarantees, the following challenges need to be addressed:

- *Guarantee QoSS requirements.* The challenge is to detect when performance will violate QoSS requirements, and determine how to avoid this via appropriate resource re-allocation. Analytical techniques are needed to identify the new resources as destination for data migration, if current resources are not capable of ensuring QoSS guarantees in the future.

- *Ensure minimal resource consumption.* The challenge is that the goal of minimizing utility cost conflicts with QoSS constraints. This goal is to reduce the resources allocated to applications when the system exhibits over-provisioning. However, too aggressive resource reclaim may hurt the performance, and violate QoSS, which is not desirable. Therefore, migrations that reduce resources must be considered together with migrations that increase resources for QoS guarantees.

- *Control migration.* The migration IOs should be controlled to enforce QoSS for the application workload when the migration is ongoing. This is achieved by finding the appropriate time to initiate migration. If the migration invocation time is not chosen carefully, the migration task may never complete due to "migration thrashing", which occurs when applications consume the entire storage bandwidth.

3.2 Single Migration Action

In this section, we analyze how a migration process from source disks, $\mathcal{S}$, to destination disks, $\mathcal{D}$, affects the workload and IO latency on each individual disk. We note that there are two exclusive sets of physical disks, with opposite change in the amount of resident data, which indicates performance outcome, during migration. We call a disk an α-$disk$ if the amount of data residing on the disk is decreasing over time, or a β-$disk$ otherwise. We enumerate three migration cases, as shown in Figure 2. For up migration, *an* α-$disk$ *is a disk* $\in \mathcal{S}$ *and* β-$disk$ *is a disk* $\in \mathcal{D} - \mathcal{S}$*; for down migration, an* α-$disk$ *is a disk* $\in \mathcal{S} - \mathcal{D}$ *and a* β-$disk$ *is a disk* $\in \mathcal{D}$.

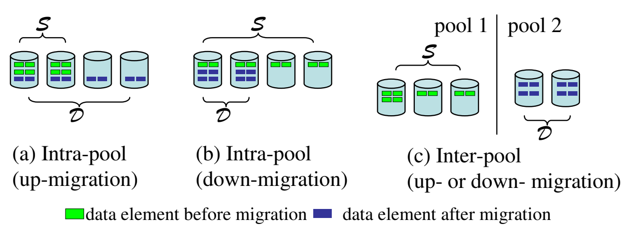


Figure 2: Cases of Migration

In order to find the appropriate time to start migration, we analyze the relationship between IO workload and migration invocation time. The fraction of I/O workload is proportional to the portion of data residing on the disk based on a uniform IO distribution assumption. In this paper, we use the workload fraction and the data fraction interchangeably. Let $\mathcal{I}$ denote the migration starting time, and t_m denote the migration duration. $p^{\alpha}(t, \mathcal{I})$ and $p^{\beta}(t, \mathcal{I})$ represent the fraction of total data in the logical disk on α-disks and β-disks respectively, at time t, and with a migration starting at time $\mathcal{I}$.

Lemma 1: Given time t, for an $\alpha\text{-}disk$, $p^{\alpha}(t, \mathcal{I})$ is non-decreasing when $\mathcal{I}$ increases, or $\frac{dp^{\alpha}(t,\mathcal{I})}{d\mathcal{I}} \geq 0$, and for a $\beta\text{-}disk$, $p^{\alpha}(t, \mathcal{I})$ is non-decreasing when $\mathcal{I}$ increases, or $\frac{dp^{\beta}(t,\mathcal{I})}{d\mathcal{I}} \leq 0$, on interval $t - t_m \leq \mathcal{I} \leq t$.

In this paper, we study one aspect of the IO workload, namely IO rate. (There are other aspects of the IO workload, such as read IO to write IO ratio, random IO to sequential IO ratio, etc.) We observed that in most cases the relationship between IO rate and IO latency can be described by the Monotonicity Assumption.

Monotonicity Assumption. During the migration process, if the application IO rate to a physical disk increases, the application IO latency increases too. Otherwise, if the application IO rate to a physical disk decreases, the application IO latency decreases too.

Based on this Monotonicity Assumption and Lemma 1, we derive that the latency function for the α-disks, referred to as $L^{\alpha}(t, \mathcal{I})$, is monotonically increasing, and the latency function for the β-disks, referred to as $L^{\beta}(t, \mathcal{I})$, is monotonically decreasing, when $\mathcal{I}$ increases.

Given the relationship between the latency function and migration invocation time, the main challenge now is to determine when to start migration so that during migration there is no QoSS violation.

Since α-disks and β-disks have different performance characteristics, we must ensure that both α-disks and β-disks do not violate the QoSS requirements during migration. Formally, we need to find a valid migration invocation time, $\mathcal{I}$, so that when $\mathcal{I} \leq t \leq \mathcal{I} + t_m$ (i.e., during the migration period), we always have $L^{\alpha}(t, \mathcal{I}) \leq L_{QoS}$ and $L^{\beta}(t, \mathcal{I}) \leq L_{QoS}$. We call the range of valid $\mathcal{I}$ a *safe time zone* for the migration, referred to as $\mathcal{RI}$.

The basic method for finding the *safe time zone* is to first calculate the safe time zone for α-disks and then generate the safe time zone for β-disks. The solution is based on two properties: (1) We utilize the monotonic property (Section 3.2) of $L^{\alpha}(t, \mathcal{I})$ and $L^{\alpha}(t, \mathcal{I})$ on $\mathcal{I}$ to enforce QoSS guarantees; (2) We only examine the latency during a migration. That is, we need to guarantee that for $t \in [\mathcal{I}, \mathcal{I} + t_m]$, there is no QoSS violation. The final *safe time zone* is simply the intersection of safe time zones of both α-disks and β-disks. Function *ActionRange* takes the migration source ($\mathcal{S}$), destination ($\mathcal{D}$), the earliest migration invocation time (t_{min}) and the latest migration invocation time (t_{max}) as input parameters and calculates the safe time zone $\mathcal{RI}$ as output.

4 Global Schedule of Migration Actions

We developed the algorithm to find the safe time zone for a single migration in the previous section. In this section, we extend the scope of the algorithm to handle multiple migrations in a sequence.

The main problem is to find a sequence of migration invocations so that no QoS violation exists in a given time frame and the resource utility is minimized. Recall that one migration invocation is defined as $(\mathcal{D}, \mathcal{I})$. We formally define the multiple migration problem in Definition 2.

Definition 2: Given discrete time points $[t_1, t_2, ..., t_n]$, and let the number of disks in the jth pool be k_j, the QoS-aware action sequence is defined as $\Gamma = \{(\mathcal{D}_1, \mathcal{I}_1), (\mathcal{D}_2, \mathcal{I}_2), ..., (\mathcal{D}_m, \mathcal{I}_m)\}$, $t_1 \leq \mathcal{I}_1 < ... < \mathcal{I}_m \leq t_n$, $1 \leq |\mathcal{D}_i| \leq k_j$ in the jth pool. Let $\mathcal{U}(\Gamma)$ denote the utility cost for sequence Γ. For each $(\mathcal{D}_i, \mathcal{I}_i)$, $1 \leq i \leq m$, there is no QoS violation. Γ is an optimal schedule iff $\forall\, \Gamma' \in$ QoS-aware action sequence, $\mathcal{U}(\Gamma) \leq \mathcal{U}(\Gamma')$.

We exploit dynamic programming to find the optimal solution and more details are presented in [8, 9]. The compuational complexity of the optimal algorithm is $O(n^3)$. As n is a measure of time, it is desirable to have an algorithm with lower compuational complexity. We now develop a sub-optimal but practical algorithm based on the above concerns. Compared to the optimal algorithm, it has linear computational cost. The intuition behind the approximate algorithm is that in the case of up migration, given $\mathcal{D}_{i-1}$, $\mathcal{D}_i$ and a migration deadline, we want to postpone the migration starting time as late as possible. This delays the increase in the utility cost. In the case of down migration, we want to start the action as early as possible, so that the utility cost is reduced at the earliest time. The ActionRange function (Section 3.3) finds appropriate migration starting time, $\mathcal{RI}$. Hence, for a single migration, the approximate algorithm picks $\max(\mathcal{RI})$ as the value of $\mathcal{I}$ for up migration, and $\min(\mathcal{RI})$ as the value of $\mathcal{I}$ for down migration. Another aspect of the approximate algorithm is when making the choice for up or down migration, the up migration is considered only when necessary, i.e. when QoS violation is expected, and the down migration is considered whenever possible.

The approximate algorithm will look forward a fixed number of N time steps. Multiple migrations are scheduled in N future time steps, and the decision on one migration will affect the later migrations. Hence, the problem of coordinating a sequence of migrations arises. Consider two migrations happening one after another, after the completion of the first migration, we may not be able to find the safe time zone for the second migration. In this case, we say that the two migrations *conflict*. The above concern motivates looking ahead into the migration sequence invoked in an IO prediction window and adjusting consecutive migrations to resolve conflicts.

5 A Case Study

In this section, we conduct a case study to examine the effectiveness and efficiency of our proposed algorithms.

5.1 Setup

We used I/O traces published by the Storage Performance Council (SPC) [13]. These traces were collected by monitoring requests to disks of an OLTP application at a large financial institution. The IO workload lasts for 2000 minutes. The IO rate varies from less than 200 IOs per minute to more than 12000 IOs per minute.

There are two pools, Pool1 and Pool2, in our storage hierarchy. Each pool contains 3 identical disks. Since we do not consider multi-pool disk assignment, there are a total of 6 disk configurations: 1) D1: using 1 disk in Pool1; 2) D2: using 2 disks in Pool1; 3) D3: using 3 disks in Pool1; 4) D4: using 1 disk in Pool2; 5) D5: using 2 disks in Pool2; 6) D6: using 3 disks in Pool2. We use two linear latency functions for disks in two pools. These two functions are collected from a storage controller (FAStT 900) with different settings. For all the disks in Pool1, their latency function is 6.6 +0.00023*IO rate (msec). For all the disks in Pool2, their latency function is 6.2 +0.00011*IO rate (msec). The first part of the latency represents the disk seek time per IO, while the second part represents the data transfer time. Considering the data transfer time, a disk in Pool2 is about twice as fast as a disk in Pool1.

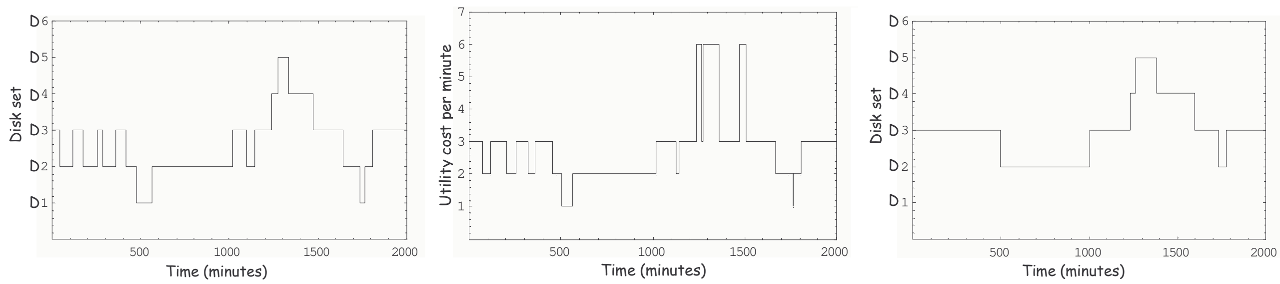
In this case study, we normalize the utility cost of each disk set over the utility cost of D1. For D1 to D3, their utility costs per minute are respectively 1, 2, 3; for D4 to D6, their utility costs per minute are respectively 3, 6, 9. This setting mimics the situation that disks in Pool2 have double the IO processing speed and triple the cost of disks in Pool1.

We set the other critical parameters with the following values: QoS latency requirement is 7.4 msec; number of migration IOs 20,000; migration duration is 30 minutes; prediction window size is 100 minutes.

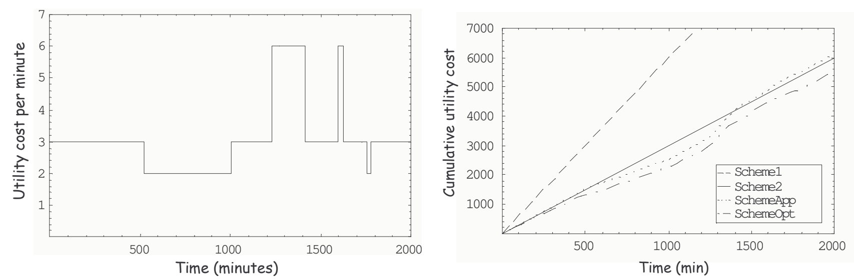
5.2 Global migration sequence

We now analyze the performance outcome of the schemes listed in Table 1 between time 0 and time 2000. Scheme1 and Scheme2 exploit static configurations, and SchemeOpt, the output of optimal dynamic programming algorithm, and SchemeApp, the output of approximate algorithm, provide dynamic configurations. We report the migration sequence and utility cost of all schemes in Figure 3.

The migration sequences generated by SchemeOpt and SchemeApp are shown in Figure 3(a) and (c). Both schemes initialize the system configuration using D3. SchemeOpt generates the migrations at a finer granularity than SchemeApp. However, SchemeApp accurately captures the trend of the big changes, although it is less responsive to small fluctuation in the workload. Specifically, SchemeApp only runs 8 migrations, less than half



(a) SchemeOpt migration sequence (b) Utility of SchemeOpt (c) SchemeApp migration sequence



(d) Utility of SchemeApp (e) Cumulative utility

Figure 3: Performance

Schemes	Policy	Meaning
Scheme1	Using Diskset5 only	Over provisioning
Scheme2	Using Diskset3 only	Average provisioning
SchemeOpt	Using the optimal algorithm	Ideal solution
SchemeApp	Using the approximate algorithm	Practical solution

Table 1: Schemes

of SchemeOpt, which dramatically reduces the amount of data movement over time. Figure 3(b) and (d) depict the utility cost per minute of the outputs for both SchemeOpt and SchemeApp. Note that the shape of the utility cost is different from that of the migration sequence because during a migration, the utility cost is the sum of the utility costs of all the disks participating in that migration. The savings in the utility cost of SchemeOpt is not significant compared with SchemeApp, although SchemeOpt aggressively moves data to minimize utility cost. Figure 3(e) shows the cumulative utility cost. The utility cost of SchemeApp grows at almost the same rate as that of SchemeOpt. The overall utility cost of SchemeApp is merely 10% more than SchemeOpt. By using slightly more utility than Scheme2, SchemeApp provides QoS guarantee. The system using Scheme2 suffers from 400 minutes QoS violation. Although the system using Scheme1 does not have any QoS violation, it incurs twice as much utility cost as SchemeApp (Figure 3(e)). In summary, SchemeApp, the approximate algorithm, efficiently minimizes the utility cost and effectively provides QoS guarantees.

6 Conclusion

PULSTORE provides an automated storage management service which balances the conflicting goals of both QoS guarantees and economical provisioning of resources. As the IO workload changes over time, storage resources need to be re-allocated to satisfy both service goals. In this paper, we developed an analytical framework to schedule data migration in large scale storage management systems. The analytical framework determines safe time zones for invoking migration actions. Using the analytical model, we then present an optimal algorithm for data migration which minimizes storage utility cost, and ensures that no QoS violation occurs. However, the optimal algorithm is predicated on complete knowledge of future workload and the computation cost is huge. Therefore, we present an approximate algorithm that schedules data migration in real time with a fixed-length IO workload prediction. The paper concludes with an analysis of real IO traces and demonstrates that the proposed approach results in significant savings in storage utility cost while preventing QoS violation effectively. More details of PULSTORE can be found in [8, 9].

References

[1] S. Chen and D. Towsley. A performance evaluation of RAID architectures. *IEEE Transactions on Computers*, 45(10):1116–1130, 1996.

[2] E. I. Cohen, G. M. King, and J. T. Brady. Storage Hierarchies . *IBM System Journal*, 28(1):62–76, 1989.

[3] IBM Corp. *AIX Logical Volume Manager from A to Z: Introduction and Concepts*. IBM Redbooks, 2000.

[4] E. Lee and R. Katz. An Analytic Performance Model of Disk Arrays. In *Proceedings of the 1993 ACM SIGMETRICS*, pages 98–109, 1993.

[5] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online data migration with performance guarantees. In *Proc. of USENIX FAST'02*, pages 219–230, 2002.

[6] S. Parekh, K. Rose, J. L. Hellerstein, S. Lightstone, M. Huras, and V. Chang. Managing the Performance Impact of Administrative Utilities. In *14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM*, pages 130–142, 2003.

[7] O. I. Pentakalos, D. A. Menasce, M. Halem, and Y. Yesha. Analytical performance modeling of hierarchical mass storage systems. *IEEE Transactions on Computers*, 46(10):1103–1118, 1997.

[8] L. Qiao, D. Agrawal, A. El Abbadi, and B. R. Iyer. Pulsatingstore: An analytic framework for automated storage management. In *International Workshop on Self-Managing Database Systems (ICDE Workshops)*, pages 12–19, 2005.

[9] L. Qiao, B. R. Iyer, D. Agrawal, and A. El Abbadi. Automated storage management with qos guarantees. In *ICDE*, pages 150–152, 2006.

[10] J. W. Ross and G. Westerman. Preparing for utility computing: The role of IT architecture and relationship management. *IBM System Journal*, 43(1):5–19, 2004.

[11] C. Ruemmler and J. Wilkes. A trace-driven analysis of disk working set sizes, HP Laboratories Technical Report HPL-OSR-93-23, April 1993.

[12] P. Scheuermann, G. Weikum, and P. Zabback. Data Partitioning and Load Balancing in Parallel Disk Systems. *VLDB Journal: Very Large Data Bases*, 7(1):48–66, 1998.

[13] Storage Performance Council:http://www.storageperformance.org.