RAIC: Scaling Storage Availability and Performance in On-Demand Data Centers

Kostas Magoutis, Murthy Devarakonda, Norbert Vogl, Kaladhar Voruganti
IBM Research

Abstract

Redundant Arrays of Independent Controllers (RAIC) is a new methodology for increasing the availability and performance of storage resources in on-demand enterprise data centers. Currently, large and dynamically-growing data sets, typical in e-business on-demand workloads, are constrained by the capacity, availability and performance boundaries of individual RAID controllers. Simple approaches to extending such data sets over multiple RAID controllers result in reduced availability guarantees. Some of the challenges in improving performance by distributing data sets over many RAID controllers include the heterogeneity in the underlying storage resources as well as the lack of information about their characteristics. RAIC is a new methodology to compose storage objects according to user-specified performance and availability goals, which may exceed the capabilities of any single individual RAID controller. RAIC policies ensure that the capacity of storage objects can be extended while maintaining performance and availability goals, even in heterogeneous environments including different types of RAID controllers.

1. Introduction

A key objective in today's e-business on-demand environments is to provide applications with uninterrupted access to a large pool of storage servers over a storage-area network (SAN). The storage servers in such environments are typically RAID-array controllers. Storage objects such as file systems or database tables are layered over virtual-disk abstractions commonly referred to as logical volumes, which in turn map to storage volumes (otherwise known as LUNs) exported from storage controllers. An example of this storage hierarchy is shown in Figure 1. Some enterprise data-centers consist of homogeneous, often identical, storage controllers. Such environments are nearly always the result of a strategic partnership between an enterprise and a storage systems vendor. Many e-business on-demand environments see such partnerships as too limiting and thus opt for the flexibility of a migration path to the vendors that offer the best price/performance benefits. This is a reason that many of today's data centers consist of heterogeneous storage resources, which can be storage controllers from different vendors or from different generations of a controller type. Cost tradeoffs as well as rapid technological advances often result in new generations of a storage controller coming into service before old generations complete their useful lifecycle.

Two important challenges in on-demand data centers today are (a) the mapping of dynamically-growing storage objects to a pool of heterogeneous storage controllers, in a scalable and seamless manner, and (b) the reduction in the complexity of managing the storage resources. In this paper, we present a methodology that addresses the first challenge. This methodology can be implemented within a storage virtualization service, which reduces the complexity of storage management by providing a common administration interface across heterogeneous storage resources.

1.1 Storage allocation models

Modern applications using a range of data-management systems, from file systems to databases, have certain performance and availability *goals* such as a certain level of I/O operations per second, mean-time-to-failure (MTTF) etc., metrics. These applications require system support for mapping their performance and availability goals to storage resources with the appropriate capabilities. A common practice is to assign each data object (e.g., a file system or a database table-space) of a storage consumer to a single storage controller as shown in Figure 1. We refer to this practice as the *one-to-one* storage allocation model. The most important benefit of the one-to-one model is its simplicity: storage characteristics such as performance and availability of a data object are directly mapped to the well-defined capabilities of a single storage controller.

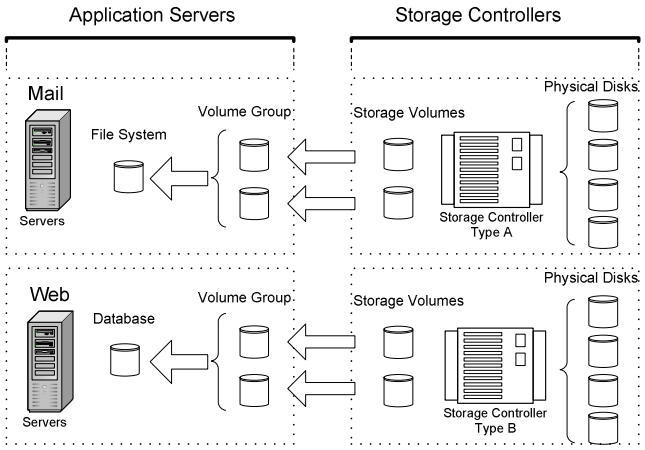

Figure 1 Storage volumes that make up a data object (e.g., a file system or a database table-space), map to a single storage controller in the one-to-one storage allocation model.

The simple one-to-one storage allocation model however, has two important drawbacks: First, a data object is constrained by the single-system limits of the storage controller (e.g. storage

capacity, performance, and availability level). This problem can be partially addressed by using *enterprise-class* storage controllers, which provide large capacity, high performance, and high availability. A problem with this approach is the *high cost* of enterprise-class storage controllers. The second drawback of the one-to-one model is that unused storage capacity in a group of storage controllers cannot be combined to create a data object. This problem of internal fragmentation of storage controller capacity leads to suboptimal storage utilization.

However, the most significant drawback of the one-to-one model is its limited scalability for data objects whose storage requirements grow, eventually exceeding the capabilities of any single RAID controller in terms of capacity, performance, or availability. A natural extension is a model that maps data objects to more than one controller, which we term the *one-to-many* model. While this model offers the potential to achieve the desired scalability characteristics, these characteristics depend on the performance and availability capabilities of *many* storage controllers. This dependence is particularly complex in the case of heterogeneous controllers. The one-to-many allocation model is not widely deployed because the only way to manage its complexity today is manually, by highly-skilled system administrators.

Besides the issue of storage allocation, another challenge in on-demand data centers is the lack of a single administrative interface across heterogeneous controllers. A single management interface is expected to reduce administration complexity and thus the total cost of ownership (TCO). A solution that promises to address this problem and has recently started to be deployed is block-level storage virtualization.

1.2 Storage Virtualization

Block-level storage virtualization [1] [2] [3] is the aggregation of storage resources into a single storage pool, managed from a single administration point. Storage virtualization provides: (a) a common administrative interface across homogeneous or heterogeneous storage resources, reducing administration costs; (b) better resource utilization, by aggregating previously unconnected storage "islands", and by consolidating all available storage volumes across storage controllers; and (c) the opportunity to implement advanced storage functions such as point-in-time copying and mirroring over all back-end storage resources, irrespective of the capabilities of the back-end controllers. Block-level storage virtualization can be implemented at various levels in a SAN: on application servers by logical volume managers (LVMs); in network elements (e.g., intelligent switches and routers [3]); or in dedicated storage virtualization controllers [2] [6], as shown in Figure 2.

An overriding concern in the design and implementation of storage virtualization engines is how to minimize the impact of virtualization on the performance (e.g., response time) and availability of back-end storage resources. In other words, the key challenge they face is to *preserve* the quality of service of the back-end storage controllers. For this reason most virtualization engines today take pains to impose minimal performance penalties, and ensure that the availability of the virtualizer itself is not lower than that of the storage controllers it exports. Techniques used to minimize the performance penalties include aggressive non-volatile caching for read and write I/O, scalable clustered designs, and lightweight implementations within network elements such as host-bus adapters (HBAs) or network switches. High availability of the virtualization service itself can generally be achieved with sufficient hardware redundancy and high-availability software features. A clustered storage virtualization controller has demonstrated such a highly-available design [2].

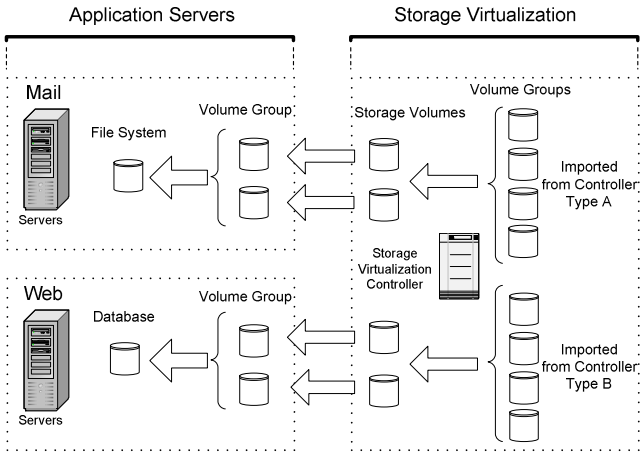

Figure 2 In this example, a storage virtualization controller imports storage volumes from storage controllers of type A and B into separate volume groups. Each storage volume exported by the virtualization controller is carved out of one of these volume groups. Note that this is equivalent to performing storage allocation using the one-to-one model.

Most deployments of storage virtualization services use the one-to-one model for storage allocation and management. This means that storage volumes from each back-end storage controller form a separate volume group, as shown in Figure 2. In this way, a volume group is available when the back-end storage controller it corresponds to is available, and the volume-group performance approximates the performance of that back-end controller. In other words, the quality of service of the back-end storage controllers is preserved. An opportunity to combine the benefits of virtualization with the *strengthening* of storage characteristics exists with stacking RAID layers in a SAN, as described in the next section.

1.3 Stacking Multiple RAID Levels in a SAN

In principle, it is possible to replace physical disks in the back-end of a RAID controller with storage volumes imported from other RAID controllers. In this way, RAID disk (or parity) groups may be composed out of combinations of heterogeneous disk resources, including physical disks as well as external storage volumes, as shown in Figure 3. We use the term *Multi-Level Heterogeneous RAID (MLH-RAID)* for this scheme. In MLH-RAID, each subsequent RAID level can be used to strengthen the performance and availability provided by its preceding RAID level. For example, to implement RAID-1 at the second level of a two-level MLH-RAID hierarchy as the one shown in Figure 3, the second-level RAID controller will mirror data blocks on two separate first-level storage volumes, which may in turn be implemented using RAID-5 over groups of physical disks at the first-level RAID controllers. There are two key differences

between MLH-RAID and a storage virtualization controller: RAID controllers are designed to *improve* on the performance and availability of back-end disks by taking advantage of disk-spindle parallelism and opportunities for data redundancy. The downside of this is that RAID controllers are expected to impose a performance penalty on the I/O data path due to the complexity of the RAID scheme. This is a fundamental tradeoff between MLH-RAID and storage virtualization controllers.

Note that there are a number of shortcomings with a simple-minded MLH-RAID implementation. First, standard RAID array implementations lack adaptivity to heterogeneous disk groups and are unaware of special availability characteristics of the underlying storage controllers. Second, storage volumes from a single storage controller are failure-dependent and should be used in different parity groups; as a result, one would need a significant number of controllers to match the desired parity-group sizes.

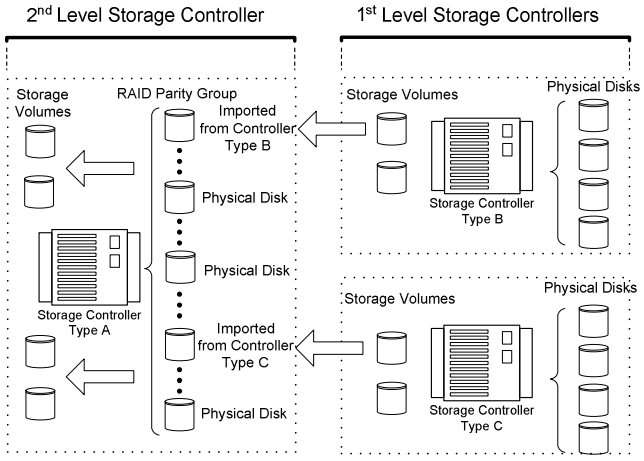

Figure 3 In a multi-layered RAID scheme, higher-level RAID controllers can create RAID disk (or parity) groups from heterogeneous storage devices, which may include direct-attached physical disk drives as well as storage volumes imported from lower-level RAID controllers.

In this paper, we emphasize that combining a storage virtualization service with an automated method of *transforming* the quality of service of a collection of heterogeneous storage resources by applying the one-to-many allocation model, is the key to achieving scalability in on-demand data centers. Two road-blocks to any such storage system implementation today are (a) the lack of complete information about the characteristics of the underlying heterogeneous storage resources, and (b) the lack of system policies to take advantage of this information. Our proposed system architecture, the *Redundant Arrays of Independent Controllers (RAIC)*, addresses these shortcomings. RAIC can be implemented as an extension to an existing storage-management system that already provides basic virtualization services. In addition, RAIC enables the composition of storage volumes that can approximate the desired availability and performance goals by combining storage resources from a sufficient number of heterogeneous RAID storage controllers. RAIC automates the management of one-to-many allocation policies by collecting knowledge about the characteristics and capabilities of the underlying RAID controllers, as well as administrator preferences, and by feeding them to a set of policies to handle events such as requests for additional storage capacity or storage controller failures, without downgrading the desired availability and performance of high-level storage objects.

Our contributions in this paper are the following:

- We describe the RAIC storage management architecture, which helps existing storage virtualization systems scale storage capacity, availability, and performance over a collection of heterogeneous controllers.
- We identify the information that RAIC needs to collect about storage resources. We also identify policies that RAIC uses to compose storage objects according to application goals and to adapt to changing conditions.
- We describe possible implementations of RAIC as part of a storage virtualization controller or in an MLH-RAID storage controller. In both cases, the management complexity is hidden behind the storage virtualization interface.

2. Related Work

Prior research on Redundant Arrays of Inexpensive[1] Disks (RAIDs) [4] [5] proposed ways to improve on the performance and availability of physical disks by laying out a virtual-disk layer over a set of physical disks and employing different forms of redundancy to tolerate the additional failure modes. Note that although the RAID architecture improves on the performance and availability of single disks, RAID controllers still face scalability limits due to single-system capacity and physical packaging constraints. Today's on-demand data centers are equipped with large numbers of generally heterogeneous RAID controllers.

The RAID abstraction offers three distinct advantages: (a) *Block-level storage virtualization, (b) disk-spindle parallelism, and (c) data redundancy.* In more detail: (a) Storage virtualization at the block-level is defined as a mapping between a set of low-level disks to a set of high-level disks, the latter sometimes referred to as storage volumes. A typical RAID controller is a storage virtualization engine which, by appropriate choice of the storage virtualization mapping, can achieve: (b) increased performance through parallelism, achieved by *striping* or *mirroring* a storage volume over more than one disk; and (c) increased availability through data redundancy. With striping or mirroring data blocks, a stream of simultaneous accesses to a storage volume can be mapped to multiple underlying disk spindles, which can operate concurrently. RAID controllers typically achieve parallelism by using block-level striping (RAID levels 0, 5) or mirroring (RAID levels 1, 10). RAID schemes for data redundancy include a variety of techniques, such as parity (RAID 5), mirroring at the block level (RAID 1 and 10), and more recently, erasure codes. Most RAID schemes to date have been designed with the assumption of homogeneous underlying storage resources. RAIC, in contrast, is designed to support adaptation in the face of heterogeneity.

The principles of the RAID architecture have been extended over network-accessible groups of disks in the Swift/RAID [7] and

[1] Because of the restrictiveness of "Inexpensive", the 'I' in RAID is sometimes said to stand for "Independent" [5].

Zebra [12] systems, which are based on distributed data-striping and parity-based availability. These projects focused primarily on scaling storage-system performance in homogeneous distributed environments. Other projects such as HERA [8] and AdaptRAID [9] considered heterogeneous environments: The HERA project proposed a RAID design over a set of heterogeneous disks by interposing a logical-disk abstraction between the two. HERA creates a set of homogeneous logical disks from a set of heterogeneous physical disks and builds the RAID abstraction on top of these logical disks. HERA considered both performance and availability of the resulting RAID abstraction; however, one drawback of the HERA approach is that it is specific to multimedia applications, which are read-mostly, throughput-sensitive and use large I/O blocks. AdaptRAID is a more general approach in that it is geared towards general-purpose workloads and scientific applications. AdaptRAID proposed a block-distribution algorithm to build a RAID disk array from heterogeneous set of disks. AdaptRAID arrays can achieve better throughput than standard RAID arrays, which typically assume that all disks have the lowest common capacity and speed. However, AdaptRAID does not consider availability tradeoffs in heterogeneous disk groups.

A number of adaptive RAID schemes explored other approaches to adapting to heterogeneity. AutoRAID [18] is a storage system that internally performs migration of data blocks between RAID level 0 (mirrored) and RAID level 5 (parity protected), trading performance for storage efficiency. AutoRAID is able to internally and dynamically migrate data when new (and possibly larger-capacity) disks are added to the array. AutoRAID is able to use all storage capacity but does not adapt to the performance and availability differential between disk drives. The WiND [19] project at Wisconsin explores a number of approaches to adapting to storage heterogeneity, such as storage-aware caching [24], exposing storage hints to the file system [26], and graceful degradation under disk failures in a RAID array controller [27].

Goal-oriented storage system design and management has been studied in the context of Hippodrome [16] and Stonehenge [15] projects among others. Hippodrome is a storage design approach that iteratively approximates a minimal-cost RAID array that is provisioned for a particular workload. Hippodrome combines trace-based workload characterization, table-based storage-device modeling, and an analytical storage-system model solver [17] based on a randomized multi-dimensional bin-packing algorithm. Stonehenge aims to enforce user-defined quality-of-service guarantees over a group of physical disks by using an approach termed measurement-based admission control (MBAC). While the general goal of Stonehenge is to guarantee QoS for multiple attributes of the virtual disks it creates, it currently focuses on performance and does not yet cover RAID schemes. RAIC is similar to Stonehenge in that it also aims to approximate multi-dimensional QoS; RAIC however, follows a simpler, more practical approach, based on storage system characterization and administrator-defined policies.

RAID arrays can theoretically achieve impressive availability levels in terms of MTTF metrics [4] [5]. However, other factors such as power outages, operator errors, scheduled down-time, etc., reduce the *operational* availability of storage controllers. To guard against controller unavailability, many enterprise and mid-range RAID controllers, and more recently, storage virtualization controllers [6], provide a storage-volume replication (or mirroring) feature, which operates as shown in Figure 4. The primary use of volume replication today is for disaster recovery, as part of a cascaded data backup scheme that replicates data to a secondary, remote site. In such a cascaded data backup scheme, a synchronous mirroring relationship or *peer-to-peer remote copy* (PPRC) operation is performed between a storage volume and its mirror within the same site; at the same time, an asynchronous replication relationship is set up between that site and a remote backup site.

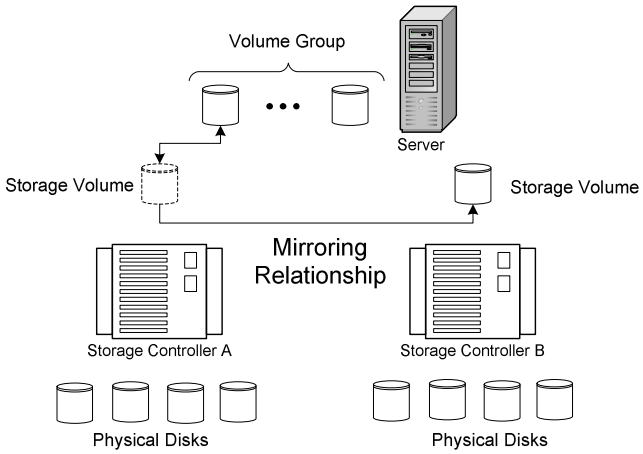

Figure 4 Volume mirroring is a data-redundancy mechanism provided at the front-end of RAID storage controllers on top of the redundancy mechanism (parity, block mirroring) that the RAID controller implements in its storage back-end. The higher-level mirroring policies (e.g., failover handling, etc.) must be specified externally by the agent that configures the mirroring relationship.

Recent research on planning for disaster recovery has focused primarily on increasing data integrity and availability to minimize financial penalties associated with data loss or unavailability [13]. An additional part of the financial cost of a solution is the cost of outlays, i.e., the cost of equipment, such as storage controllers and channel extenders. This work is closely related to RAIC; our focus, however, is in *partial* data center failures such as temporary storage controller outages, which are potentially more frequent and less severe than entire-site failures. Use of volume replication to survive controller failures within a single site has not been studied in depth and management software that makes extensive use of such a facility is not an integral part of on-demand infrastructure processes today.

Automated availability management in a heterogeneous setting has recently been studied in the context of peer-to-peer systems [14]. This work is related to ours in that storage consumers can specify availability goals that the system tries to achieve by using redundancy techniques (replication and/or erasure coding) on top of the measured availability characteristics of individual hosts that are storing data. This work, however, does not take into account multiple storage attributes (e.g., performance in addition to availability), it uses availability measures that are more appropriate for the Internet, and is geared towards environments

with significant "churn", i.e., high frequency of storage hosts entering and departing the system.

Brown and Patterson [10] made the case for characterizing system availability using a benchmarking methodology. In their approach, they considered availability as a spectrum rather than a simple binary metric ("up" or "down") or even an average of the percentage of time that a system is available. They took into account various states of degraded performance and rejected the notion that availability can be defined at a point in time or as a simple average over time. Instead, they propose examining the variations in a system's quality of service over time, where the notion of quality of service varies depending on the type of system studied. For storage systems, as well as for most servers, performance (e.g., IOs/sec) and degree of fault tolerance (e.g., number of failures that can be tolerated) are two obvious metrics. Brown and Patterson proposed taking availability measurements while injecting one or more faults and using graphs and numerical summaries of these time-dependent measurements as system-availability characterizations.

Existing high-availability specifications such as IBM's Highly-Available Cluster Multi-Processing (HACMP) [21] require either volume mirroring at the LVM level or RAID data redundancy within a storage controller to achieve high availability. However, for large data sets that have to span multiple controllers, possibly of different characteristics and capabilities, a new methodology that provides stronger availability guarantees is required. RAIC is positioned to fill this need.

3. Redundant Arrays of Independent Controllers

Redundant Arrays of Independent Controllers, or RAIC, is a new storage management architecture that can approximate user-specified capacity, performance, and availability goals in on-demand data centers. RAIC distributes storage objects over groups of underlying storage volumes from one or more heterogeneous back-end storage controllers. Key tasks of RAIC are the composition of storage resources from groups of heterogeneous storage volumes, and the availability and performance management of these storage resources. RAIC reacts to dynamic conditions such as requests for additional storage capacity extensions, or environmental changes, such as transient or long-term failures of RAID controllers. On each such action, RAIC policies take into account knowledge about the current state of the system. For example, detailed on-site classification of the availability of RAIC controllers, as described in Section 3.2, is critical in achieving availability goals and rapidly recovering from planned or accidental controller down-time. RAIC policies can also include administrator beliefs in the dependability of on-site storage controllers.

The central component of RAIC, whose architecture is depicted in Figure 5, is the Volume Manager (VM). The RAIC VM is responsible for creating RAIC Volume Groups (VGs), which are collections of storage volumes imported from back-end RAID controllers and associated with certain capacity, availability and performance characteristics [22] [23]. Storage consumers that use RAIC as a storage manager, import storage volumes carved out of RAIC VGs. The RAIC VM contains policies to create and extend a VG, and to proactively or reactively handle dynamic state changes such as transient or long-term underlying storage volume unavailability.

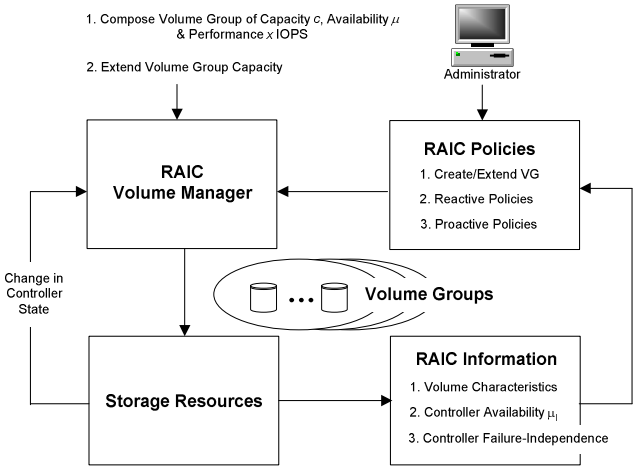

Figure 5 The RAIC VM composes or extends VGs according to capacity, performance, and availability goals. RAIC responds to changes in controller-availability state using availability and performance classification of RAID controllers as well as administrator input.

The underlying storage volumes managed by RAIC are characterized by their storage capacity and the identity of their respective back-end storage controllers. Ideally, these storage volumes would be associated with specific performance and availability characteristics, as is currently the case with disk drives. Such guarantees, however, are hard to achieve for storage volumes exported from RAID controllers. To account for this lack of availability and performance guarantees, the RAIC VM collects and maintains information about the underlying storage volumes and the back-end controllers that export them. This information is used along with RAIC policies to best approximate the capacity, availability, and performance characteristics associated with the RAIC VGs. The information collected and maintained by the RAIC VM, as shown in Figure 6, consists of capacity and implementation characteristics of the underlying storage volumes; performance and availability characterization of the back-end storage controllers; and the level of failure-independence between storage controllers.

3.1 Storage Volume Characteristics

The RAIC VM collects information from back-end storage controllers regarding the capacity, implementation and access paths of storage volumes. This information is used to determine whether any two storage volumes contend for resources, such as physical-disk spindles (e.g., if they are carved out of the same RAID disk group on the same controller) or data paths through a single controller or through the storage-area network. The lower the level of contention between two storage volumes, the closer these storage volumes are to being branded *performance isolated* [25]. This information becomes particularly important when it is critical to achieve the performance goals in a VG. Some of this information can be collected through storage management interfaces such as SMI-S [28].

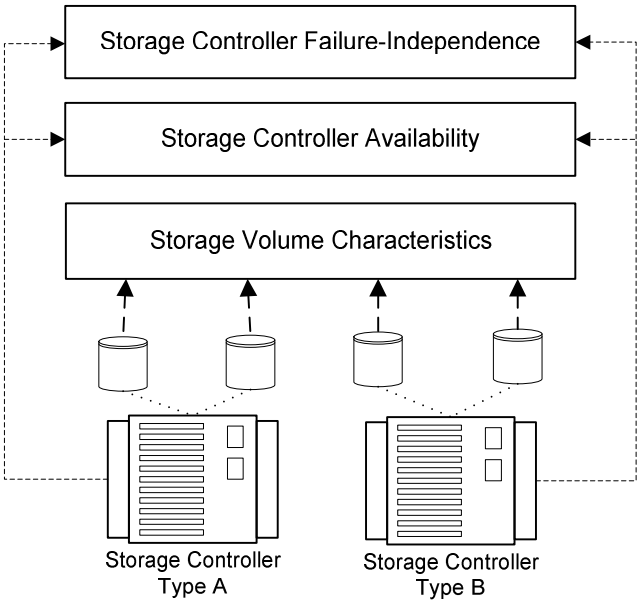

Figure 6 Information about storage controllers and storage volumes collected and maintained by RAIC.

3.2 Storage Controller Availability

The RAIC VM collects and maintains information about back-end storage controller availability. This information is frequently updated automatically and also through administrator input, to reflect the administrator's experience and confidence of back-end storage controller's operating behavior. Most of today's RAID controllers that are classified as Highly Available (HA) are built using redundant hardware components to sustain failure of any single hardware component. Their effective availability, however, depends significantly on other factors, such as the design and behavior of their systems management software with respect to transient errors, its recovery policies, environmental factors such as installation issues, operator errors, etc. Our empirical evidence suggests that the intervals of unavailability of highly-available controllers as a side effect of operator errors or simply during heavyweight administrative tasks are not negligible. In addition, some data centers include mid-range or low-end RAID controllers that do not offer a high degree of hardware redundancy and as such, certain types of failures such as transient faults can render such controllers inaccessible.

The availability of RAID controllers depends in part upon the reliability of the SAN fabric, the controller hardware and software components, and its disk drives [5]. Disk drives typically operate for long time intervals before failing (MTTFs of modern disks are in the order of 100,000 hours). When they fail, they are almost always replaced rather than repaired. Sometimes a "stutter" time interval precedes failure [20]. Storage controllers on the other hand, after experiencing a period of unavailability, they are nearly always repaired rather than replaced. Moreover, controller down-time is not always due to component failures. Periodic maintenance and other corrective and preventive actions contribute to controller down-time, making *operational availability* more appropriate as a measure of controller availability, as expressed in the following equation:

Availability = MTBM / (MTBM + MDT).

The history of a storage controller's *mean-time-between-maintenance* (MTBM) and *mean downtime* (MDT) is a good indicator of its future availability behavior. This information can be useful and used in addition to MTTF, MTBF, and MTTR metrics that are usually published by the storage controller vendor. Finally, additional input in the form of administrator beliefs about a controller's availability, which take into account environmental factors such installation issues, lack of power-line redundancy, etc., are also part of a storage controller's characterization.

For the purpose of this paper, we suggest a discrete availability classification (μ) of storage controllers and leave more details of this characterization for future work. This availability scale ranges from 0 to 1, as shown in Figure 7. Note two important points about this availability classification: First, the mapping of a RAID storage controller to a certain availability level is *dynamic* and *periodically re-evaluated*; this characterization may change based on current conditions such as confidence on the systems management software and operator experience. Second, certain high availability levels (e.g., $\mu = 0.99$) may not be achievable by any single RAID controller alone, but only through an external management system such as RAIC.

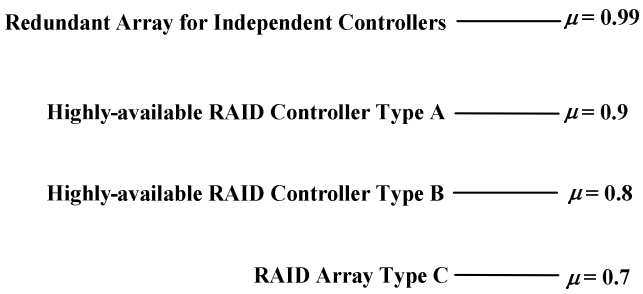

Figure 7 A storage controller can dynamically move between availability levels based on its history and current confidence on its operating parameters. Achieving arbitrarily high levels of availability (such as μ = 0.99) may only be possible with techniques such as those used by RAIC.

3.3 Storage Controller Failure-Independence

The RAIC VM collects failure-independence measurements between storage controllers. When including storage volumes from several back-end storage controllers in a RAIC VG, the overall availability of the VG depends on the failure-independence of the back-end storage controllers. Standard RAID-array availability calculations make the assumption that back-end disk drives operate with independent failure modes [5]. This assumption does not always hold for storage volumes carved out of different storage controllers in typical data centers. For example, in many cases, groups of storage controllers are powered from the same power line(s) or are accessible from the same SAN switch, etc. Such storage controllers (and therefore all storage volumes exported from them) are failure-dependent. RAIC

collects information from administrators about the failure-dependency of storage controllers and uses it in availability calculations of VGs.

4. RAIC Implementations

In this paper we describe two possible RAIC implementations. The first implementation extends the MLH-RAID scheme described in Section 1.3. In such a setup, the RAIC functionality adds information and policies, necessary to handle storage volumes from multiple heterogeneous controllers, to a simple-minded RAID implementation. For example, additional data redundancy and stripping over lower-quality disks as well as storage-aware caching [24] can improve the overall performance and availability of a heterogeneous parity group. In the second implementation, RAIC is integrated with a storage virtualization controller, using externally-implemented mirroring operations to strengthen data availability guarantees. In the remainder of the paper, we will refer to such an implementation as RAIC-SVC. We believe that both implementation options are practically viable and equally promising. Due to space limitations, in this paper we focus on the second implementation.

A RAIC-SVC Volume Group is composed of two sets of underlying storage volumes: an *Active Set* and a *Mirror Set*. The Active Set contains storage volumes from one or more back-end storage controllers, chosen based on the capacity and performance goals. The Mirror Set contains storage volumes from one or more back-end storage controllers that mirror selected storage volumes from the Active Set. The storage volumes in the Mirror Set are chosen based on the data availability goals. The RAIC VM is careful to mirror volume across controllers with independent failure modes. Data accesses are performed on the Active Set and updates are reflected to the Mirror Set via the mirroring relationships set up by RAIC. In the event of controller failure(s), RAIC maintains data access by failing over to the surviving storage volume(s).

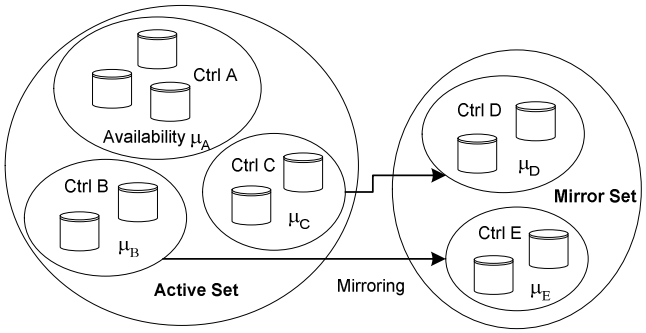

Figure 8 An example of a RAIC-SVC VG configuration.

Note that RAIC separates the management of storage capacity and performance (i.e., the composition of the Active Set), from the management of storage object availability (i.e., the composition of the Mirror Set and the replication relationships between the Active and Mirror Sets). In the remainder of this paper we focus primarily on the management of availability. One reason behind our choice is that there already exists previous research addressing the issue of performance over heterogeneous storage volumes [8] [9] [19]. Another reason is a prevailing belief in the systems community that availability planning and recovery-oriented computing are a key priority in systems research today [10] [11].

Next, we describe the policies used in creating and extending RAIC-SVC VGs, and in proactively or reactively responding to events that signal a change in the state of a VG.

4.1 Creating a RAIC-SVC Volume Group

When creating volume groups, RAIC-SVC first considers whether striping over multiple underlying storage volumes is necessary. This would be the case if the overall performance goal of the VG exceeds the capabilities of any single back-end RAID controller; if so, RAIC-SVC will stripe data over multiple volumes. It is possible that striping may be necessary on only a subset of the volumes comprising the VG. This subset may be comprised of lower-performance volumes in a heterogeneous VG. An important issue is the unit of striping; RAIC will decide on the stripe size taking into account application workload characteristics.

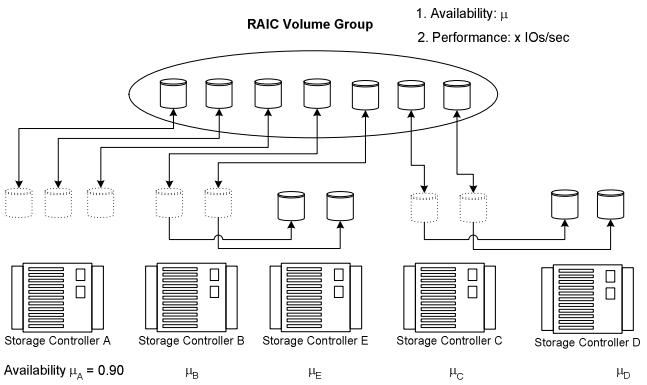

Figure 9 A realization of the VG configuration of Figure 8.

RAIC-SVC uses the following policies when creating a volume group: Initially, it attempts to find a single back-end storage controller that satisfies *both* the performance and availability goals of the VG. The reason for doing this is to be able to get as close as possible to the performance goal with minimal availability loss. If a single storage controller can be found, RAIC-SVC tries to allocate all underlying storage volumes in the VG from that storage controller (greedy approach). RAIC-SVC also uses guidance policies reflecting business or other goals in selecting a suitable storage controller. If there is no single storage controller with the desired capacity, RAIC-SVC looks for multiple storage controllers with similar characteristics. When performance is critical in a VG, RAIC-SVC chooses to allocate performance-isolated storage volumes (Section 3.1).

After it allocates a number of storage volumes that can satisfy the capacity and approximate the performance goals of the VG, it adds these volumes to the Active Set. RAIC-SVC then proceeds to choose a number of storage volumes for the Mirror Set and set up a sufficient number of mirroring relationships to achieve the availability goals of the VG. Mirroring will be necessary for two reasons: (a) a RAIC VG may contain volumes from a single back-

end RAID controller but may have an availability goal that exceeds the capabilities of the controller, or (b) a RAIC VG may contain storage volumes from several back-end RAID controllers.

4.1.1 Examples

Next, we will provide examples where mirroring helps achieve the availability goals of a data set even as the latter grows dynamically to incorporate storage volumes spanning heterogeneous (and sometimes lower quality) storage arrays. In all examples we assume failure-independence between all controllers.

In the example of Figure 10, we assume that the availability of each individual RAID array is $\mu = 0.9$ (this can be interpreted as meaning "the array is available 90% of the time") but the storage consumer has set an availability goal of 0.99. By mirroring each volume of RAID controller A to a volume on RAID controller B, the overall availability of the system is improved to 0.99 as the following calculation shows:

Availability = **Pr**[RAID Array A **or** RAID Array B available] =

= 1 – **Pr**[RAID Array A **&** RAID Array B unavailable] =

$= 1 - (1 - \mu)^2$.

For $\mu = 0.9$, the overall availability matches our goal of 0.99. The above example demonstrates the case where a RAIC-exported storage volume can achieve higher availability than any of the underlying storage controllers are individually capable of providing.

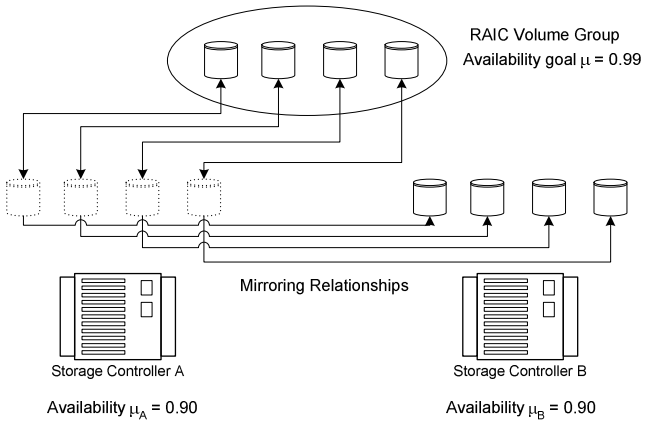

Figure 10 Strengthening availability by mirroring across storage RAID array controllers. In this example, the two storage controllers are equivalent and characterized by availability μ. Note that RAIC provides the failover capability required in case of failure of RAID controller A.

In the example of Figure 11 we show how a storage object which is allocated or dynamically extended using storage volumes of different capabilities can compensate for differences in the availability characteristics of its subcomponents. In this example, RAIC starts allocating storage volumes for a volume group from RAID controller A, which matches the VG availability goal of $\mu = 0.90$. However, as RAID controller A eventually runs out of space, a subsequent capacity extension request will have to allocate storage volumes from RAID controller B, which offers a lower availability level of $\mu_B = 0.80$. To be able to match the availability goal of $\mu = 0.90$, RAIC will decide to mirror each storage volume of RAID controllers A and B to storage volumes on RAID controllers C and D, respectively. In the resulting RAID controller pairs AC and BD, it is sufficient for a single controller in a pair to be available for the storage volumes in the pair to be available. The improved availability of the system is calculated as follows:

Availability = **Pr**[RAID Pair AC **&** RAID Pair BD available] =

= 1 – **Pr**[RAID Pair AC **or** RAID Array Pair BD unavailable] =

$= 1 - [(1 - \mu_A)(1 - \mu_C) +$

$+ (1 - \mu_B)(1 - \mu_D) - (1 - \mu_A)(1 - \mu_B)(1 - \mu_C)(1 - \mu_D)]$.

For the values of μ_A=0.90, μ_B=0.80, μ_C=0.80, μ_D=0.80 in this example, the overall availability is 0.948, which achieves and slightly exceeds our goal of 0.90.

In the general case, when formulas calculating effective availability cannot be easily inverted, RAIC-SVC will enumerate the effective availability resulting out of using a group of RAID controllers of different characteristics (μ_i) and will decide on the appropriate level of redundancy needed by inspection of the availability (A-μ_i) point graph.

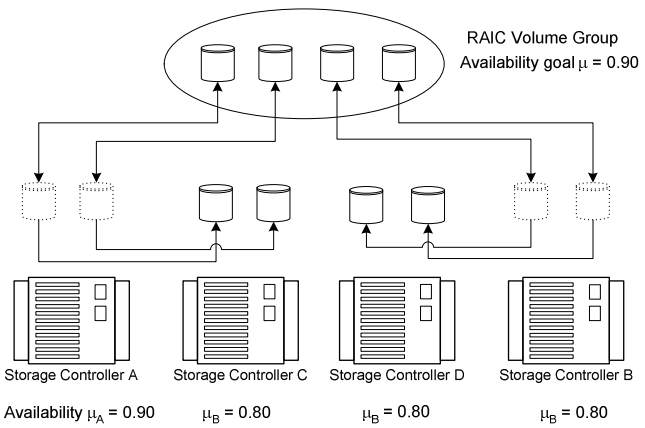

Figure 11 Maintaining availability goals for a RAIC volume group as the latter grows and incorporates storage volumes across RAID controllers of varying capabilities. In this example, primary RAID controllers A and B are characterized by availability levels μ_A and μ_B, respectively. Secondary RAID controllers C and D are characterized by availability levels μ_C and μ_D, respectively.

4.2 Extending a RAIC-SVC Volume Group

When extending a volume group, as in the initial-allocation policies, RAIC-SVC picks a storage volume that is as close as

possible in terms of capabilities to the storage volumes already in the VG. If lower-performance volumes are included in the VG, additional striping over those volumes may be necessary to improve performance through spindle parallelism. Similar to the initial-allocation case, mirroring may be necessary to improve availability guarantees.

4.3 Reacting to Dynamic State Changes

One of the key tasks of the RAIC-SVC VM is to establish and maintain the state of a set of volume replication relationships between underlying storage volumes. It does so transparently to applications that use the high-level storage volumes exported by RAIC. To achieve this, RAIC is able to respond to a number of events that signal a change in the state of a replication relationship between underlying storage volumes in back-end RAID controllers. In particular, RAIC policies respond to indications of storage volume unavailability by taking into account up-to-date information about the state of each back-end controller; RAIC dynamically characterizes failures as transient or permanent and adjusts its response accordingly.

Next, we give an example of how RAIC-SVC responds to the following events: (a) *Failure of the primary volume in a replication relationship***:** If RAIC-SVC does not expect rapid recovery of the primary volume, it will break the replication relationship; it will then use the secondary volume (mirror) as the new primary, allocate a new secondary volume, and establish a new replication relationship. If however it expects that the primary volume will recover soon (based on past history or hints provided by a systems administrator), RAIC-SVC will failover to the secondary volume (to quickly resume operations) but will not allocate a new secondary volume; instead, it will wait for the previous primary to re-appear. When this happens, it will re-establish the mirroring relationship in the reverse direction. This is expected to significantly reduce the overhead of synchronizing the two volumes. Another event handled by RAIC-SVC is the (b) *failure of a secondary volume in a replication relationship*: This event signals temporary reduction of the availability guarantees of the VG; in this case, just like in case (a), RAIC-SVC will either wait for the secondary volume to re-appear or it will proceed with setting up a new replication relationship to a new secondary volume. The tradeoff in both cases is between the duration of the downgraded availability time window vs. the overhead of setting up a new replication relationship and synchronizing the two volumes, which may require significant data movement.

RAIC can handle other dynamic state-change events, in addition to primary and secondary storage-volume failures. More details are left for future work.

4.4 Proactive Actions

Besides handling dynamic events that require an immediate response, such as controller failures, RAIC includes policies that proactively modify the state of the system in anticipation of such events. One example is taking early failover action or reversing a replication relationship when a storage controller hosting one or more primary storage volumes is deemed to be imminently failing or scheduled to soon be taken out of service. RAIC decides that a controller's availability is declining either by consulting with measurements showing that its performance is degrading (a sign of imminent failure, similar to disk drives [20]) or by explicitly being informed by an administrator through policy. A key benefit of proactive action is significant overall reduction in recovery time.

5. Conclusions

In this paper we argue that, in today's on-demand data centers, there is a need to scale the storage capacity, availability, and performance of data objects (e.g., file systems or database table-spaces) beyond the boundaries of a single storage controller. Distributing data objects over multiple controllers requires use of data-redundancy techniques to account for the additional failure modes. A problem with achieving such data distribution is the lack of information, such as the characteristics of storage volumes, the expected availability of storage controllers, and the failure-independence between storage controllers. Our proposed system architecture, the Redundant Arrays of Independent Controllers (RAIC), collects this information through a variety of sources: storage administration interfaces; past history of controller availability; administrator input about installation and planning issues that can affect future availability and dependence on controllers. This information is used by RAIC to approximate availability and performance goals of application data objects. In addition, it is used by RAIC policies to adapt to dynamic state-changes, such as scheduled or accidental storage controller down-time. We are currently in the process of implementing a RAIC prototype and plan to report our experience in a future paper.

6. REFERENCES

[1] E. Lee, C. Thekkath, "Petal: Distributed Virtual Disks", in *Proc. of 7Ith International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Cambridge, MA, October 1996.

[2] J. S. Glider, C. F. Fuente, W. J. Scales, "The Software Architecture of a SAN Storage Control System", in *IBM Systems Journal*, 42(2):232-249, 2003.

[3] HP Storage Virtualization Strategy, ftp://ftp.compaq.com/pub/products/storageworks/HPVirtualizationStrategy.pdf

[4] D. Patterson, G. Gibson, and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)", in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, Chicago, IL, June 1988.

[5] P. Chen, E. Lee, G. Gibson, R. Katz, D. Patterson, "RAID: High-Performance, Reliable Secondary Storage", in *ACM Computing Surveys*, 26(2):145–185, June 1994

[6] IBM TotalStorage SAN Volume Controller and SAN Integration Server, http://www.redbooks.ibm.com/redbooks/pdfs/sg246423.pdf

[7] D.D.E. Long, B. R. Montague, and L. F. Cabrera, "Swift/RAID: A distributed RAID system", in *Computing Systems*, 3(4), Summer 1994.

[8] R. Zimmermann, S. Ghandeharizadeh, "HERA: Heterogeneous Extension to RAID", in *Proc. of 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, NV, 2000.

[9] T. Cortes, J. Labarta, "Extending Heterogeneity to RAID Level 5", in *Proc. of 2001 USENIX Annual Technical Conference*, Boston, MA, June 2001.

[10] A. Brown and D. Patterson, *"Towards Availability Benchmarks: A Case Study of Software RAID Systems"*, in *Proc. of USENIX Annual Technical Conference*, San Diego, CA, June 2000.

[11] D. Patterson, and others, "Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies", in *UC Berkeley CS Technical Report UCB//CSD-02-1175*, March 15, 2002.

[12] J. Hartman, J. Ousterhout, "The Zebra Striped Network File System", in *ACM Transactions on Computer Systems (TOCS)***,** 13(3):274-310, August 1995.

[13] K. Keeton, C. Santos, D. Beyer, J. Chase, J. Wilkes, *"Designing for Disasters"*, in *Proc. of USENIX Conference on File and Storage Technologies (FAST'04)*, San Francisco, CA, March 2004.

[14] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, G. Voelker, "Total Recall: System Support for Automated Availability Management", in *Proc. of USENIX Conference on Networked Systems Design and Implementations'04*, San Francisco, CA, March 2004.

[15] L. Huang, G. Peng, T. Chiueh, "Multi-Dimensional Storage Virtualization", in *Proc. of SIGMETRICS – PERFORMANCE 2004*, New York, NY, June 2004.

[16] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, A. Veitch, "Hippodrome: Running Circles Around Storage Administration", in *Proc. of Conference on File and Storage Technologies (FAST'02)*, Monterey, CA, January 2002.

[17] E. Anderson, M. Kallahalla, S. Spence, R. Swaminathan, and Q. Wang, "Ergastulum: Quickly Finding Near-Optimal Storage System Designs", *HP Laboratories SSP Technical Report HPL-SSP-2001-05*, June 2002.

[18] J. Wilkes, R. Golding, C. Staelin, T. Sullivan, "The HP AutoRAID Hierarchical Storage System", in *ACM Transactions on Computer Systems (TOCS)*, 14(1):108-136, February 1996.

[19] A. Arpaci-Dusseau, R. Arpaci-Dusseau, J. Bent, B. Forney, F. Popovici, S. Muthukrishnan, O. Zaki, "Manageable Storage via Adaptation in WiND", in *Proc. of the 2001 IEEE Symposium on Cluster Computing and the GRID (CCGrid'01)*, Brisbane, Australia, May 2001.

[20] R. Arpaci-Dusseau and A. Arpaci-Dusseau, "Fail-stutter Fault Tolerance", In *Proc. of Workshop on Hot Topics in Operating Systems*, Schloss Elmay, Germany, May 2001.

[21] AIX High Availability/Cluster Multi-Processing Specification, Planning and Installation Guide, Document No. SC23-4861-0

[22] J. Wilkes, "Traveling to Rome: QoS Specifications for Automated Storage System Management", in *Proc. of International Workshop on Quality of Service (IWQoS-2001)*, Karlsruhe, Germany, June 2001.

[23] M. Devarakonda, D. Chess, I. Whalley, A. Segal, P. Goyal, A. Sachedina, K. Romanufa, E. Lassettre, W. Tetzlaff, W. Arnold, "Policy-Based Autonomic Storage Allocation", in *Proc. of 14th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management (DSOM 2003): Self-Managing Systems*, Heidelberg, Germany, October 2003.

[24] B. Forney, A. Arpaci-Dusseau, R. Arpaci-Dusseau, "Storage-aware Caching: Revisiting Caching for Heterogeneous Storage Systems", in *Proc. of First USENIX Conference on File and Storage Technologies (FAST 2002)*, Monterey, CA, January 2002.

[25] C. Lumb, A. Merchant, G. Alvarez, "Façade: Virtual Storage Devices with Performance Guarantees", in *Proc. of Second USENIX File and Storage Conference (FAST)*, San Francisco, CA, March 2003.

[26] T. Denehy, A. Arpaci-Dusseau, R. Arpaci-Dusseau, "Bridging the Information Gap in Storage Protocol Stacks", in *Proc. of 2002 USENIX Annual Technical Conference*, Monterey, CA, June 2002.

[27] M. Sivathanu, V. Prabhakaran, A. Arpaci-Dusseau, R. Arpaci-Dusseau, "Improving Storage System Availability with D-GRAID", in *Proc. of Third USENIX Conference on File and Storage Technologies (FAST 2004)*, San Francisco, CA, March 2004.

[28] Systems Management Interface-Storage (SMI-S) Specification, v.1.0.2, http://www.snia.org/smi/tech_activities/smi_spec_pr/spec/SMIS_1_0_2_final.pdf