ThriftStore: Finessing Reliability Trade-Offs in Replicated Storage Systems

Abdullah Gharaibeh, Samer Al-Kiswany, *Student Member*, *IEEE*, and Matei Ripeanu

Abstract—This paper explores the feasibility of a storage architecture that offers the reliability and access performance characteristics of a high-end system, yet is cost-efficient. We propose ThriftStore, a storage architecture that integrates two types of components: volatile, aggregated storage and dedicated, yet low-bandwidth durable storage. On the one hand, the durable storage forms a back end that enables the system to restore the data the volatile nodes may lose. On the other hand, the volatile nodes provide a high-throughput front-end. Although integrating these components has the potential to offer a unique combination of high throughput and durability at a low cost, a number of concerns need to be addressed to architect and correctly provision the system. To this end, we develop analytical and simulation-based tools to evaluate the impact of system characteristics (e.g., bandwidth limitations on the durable and the volatile nodes) and design choices (e.g., the replica placement scheme) on data availability and the associated system costs (e.g., maintenance traffic). Moreover, to demonstrate the high-throughput properties of the proposed architecture, we prototype a GridFTP server based on ThriftStore. Our evaluation demonstrates an impressive, up to 800 Mbps transfer throughput for the new GridFTP service.

Index Terms—Distributed storage, modeling storage reliability trade-offs, low-cost storage.

1 Introduction

A number of existing distributed storage systems (e.g., cluster-based [1], [2] and peer-to-peer [3], [4] storage systems) aim to offer reliable, cost-effective data stores on top of commodity or even donated storage components. To tolerate failure of individual nodes, these systems use data redundancy through replication or erasure coding. These systems, however, face two challenges: first, regardless of the redundancy level used, there is always a nonzero probability of a burst of correlated permanent failures that may lead to permanent data loss. Second, provisioning these systems to guarantee user-defined levels of service is challenging due to the complex relationships among the characteristics of the system components (storage nodes and network interconnect), data placement decisions, failure characteristics, and overall system properties like data reliability and throughput.

This study addresses these challenges in two ways: first, it explores the feasibility of a hybrid storage architecture, which we dub ThriftStore from "thrifty storage," that offers the data access throughput and reliability of enterprise class systems. Second, it presents analytical- and simulation-based tools to support system provisioning and performance analysis.

To obtain high I/O throughput and strong durability while keeping cost under control, ThriftStore integrates two types of low-cost storage components. First, *durable nodes*: a set of dedicated, yet low-throughput nodes that provide data durability. For example, this component can be an Automated Tape Library (ATL), which, compared to commodity disks, has an order of magnitude longer mean time to failure, is two to three times cheaper in terms of total cost of ownership, and consumes almost one order of magnitude less power [5]. Second, *volatile nodes*: a large number of unreliable nodes that, on aggregate, provide a high throughput front-end data store. For example, these nodes can be a subset of the desktops available in a company or a research institution whose idle storage is opportunistically scavenged.

One usage scenario for this hybrid architecture is a GridFTP server [6]. To enable the high I/O access rates required by scientific applications, existing GridFTP deployments are often supported by dedicated hardware resources (e.g., parallel file systems running on clusters). Integrating GridFTP with a combination of dedicated and scavenged storage resources offers the opportunity to lower the deployment cost while maintaining high performance. Another usage scenario is a high-performance data-store geared toward a read-mostly workload that uses a storage utility (e.g., Amazon's S3 [7]) for data durability, and local or cloud-provided workstations as the front-end for high-performance data access. For instance, such storage service can be used to support large-scale photo-sharing web services (e.g., Flickr [8]) which serves a read-dominant workload and requires high durability. Finally, we note that ThriftStore's two-tier architecture naturally supports many-task computing (MTC) applications as, typically, these applications are data intensive [9] and require that data be made available to a large set of computing nodes. In an MTC scenario, ThriftStore's front-end layer is built by aggregating the underutilized storage space of the allocated compute nodes and will shield the shared systemwide storage system (which will stand as the persistent ThriftStore back end) from most of the read load generated by the MTC application. As a matter of fact, recent research projects recommend the proposed architecture [10], [11], yet without integrating the various storage resources.

ThriftStore's hybrid architecture enables complete decoupling of the two components of data reliability: *durability* (ability to preserve the data) and *availability* (ability to serve the data when requested). For many applications, durability

• *The authors are with the Electrical and Computer Engineering Department, The University of British Columbia, 2332 Main Mall, Vancouver, BC V6T 1Z4, Canada. E-mail: abdullah@ece.ubc.ca.*

Manuscript received 3 Dec. 2009; revised 18 June 2010; accepted 27 July 2010; published online 23 Aug. 2010.
Recommended for acceptance by I. Raicu, I. Foster, and Y. Zhao.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDSSI-2009-12-0599.
Digital Object Identifier no. 10.1109/TPDS.2010.157.

is the critical requirement: applications may tolerate short-term storage service interruptions (i.e., lower availability) if data are not permanently lost. Decoupling durability and availability offers an opportunity to engineer systems that provide strong durability guarantees while relaxing the availability guarantees to control the overall cost.

ThriftStore's hybrid architecture brings, however, two challenges. The first challenge is to correctly provision the controlled resources to provide user-defined performance levels. In this architecture, the scarce resource is the bandwidth of the durable component(s), which may limit data availability in the entire system. Other characteristics that affect availability include the data volume (number of objects and their size), the replication level, the characteristics of the volatile nodes (e.g., failure characteristics, network capabilities, and storage space) and the characteristics of the interconnection network. In order to design tools that guide resource provisioning to provide user-defined availability guarantees, we explore the relationship between the above system characteristics and data availability.

The second challenge is to design an efficient data placement and replication scheme that generates low I/O demand on the durable nodes while capitalizing on the bandwidth of the volatile, high-throughput nodes. This requirement is important to maximize systemwide availability and to minimize the amount of data that need to be restored using the durable, low-bandwidth component.

To tackle these challenges, we address the following more concrete questions:

- Consider a system built atop of volatile, unreliable components only, which employs replication to provide both durability and availability. *How does the data loss probability relate to various system parameters (e.g., data volume, repair bandwidth, and replication level)*?
- In the symmetric system considered above, we introduce a low-bandwidth yet durable component (e.g., a tape). We assume that each object has one replica stored on this component yet, due to its low bandwidth, clients cannot directly fetch the data objects stored (i.e., replicas must exist on the high-bandwidth components to be available to clients). This component, in effect, provides durability but not availability. We are interested in *estimating the availability loss resulting from having one replica stored on a medium with low access rate*.
- Having integrated the durable component, and given the characteristics of the volatile nodes (e.g., mean time to failure and maximum bandwidth allocated to repair traffic), *what is the impact of resource constraints (e.g., bandwidth at the durable and volatile nodes) on the resulting availability level and, consequently, what is the volume of repair traffic generated at the durable and volatile nodes*?
- Once the system and workload characteristics are fixed (e.g., bandwidth constraints and failure rates), *what replication and replica placement scheme enable maximum availability*?

To answer the above questions, this paper presents an analytical model based on Markov chains. The model captures the key relationships between availability and the main system characteristics: the failure characteristics of the volatile nodes, and the maintenance bandwidth allocated at the volatile and durable components. Further, to study the impact of other factors that cannot be included in the analytical model, as they would make it intractable (such as the details of replica placement scheme and those of the deployment environment); we develop a low-level simulator of the proposed architecture. The simulator uses as inputs machine availability traces, implements the behavior of the replication scheme used, and enables an accurate evaluation of the system's characteristics.

Analytical modeling and simulation, however, do not shed any light on the practical challenges of implementing the proposed architecture and or on its throughput characteristics. To shed light on these issues, we developed a proof-of-concept prototype based on the GridFTP usage scenario. The prototype integrates the Globus' project GridFTP server [12] and MosaStore [13], a data store based on scavenged storage.

The contributions of this work are summarized as follows:

- First, we propose a low-cost, reliable, high-performance data-store architecture. We explore the trade-offs enabled by the proposed solution, and compare it to a pure replication-based solution using analytical modeling and simulations (Section 3 discusses our system model while Section 4.3 presents the evaluation results).
- Second, we provide tools that allow for dimensional analysis and system provisioning: an analytical model that captures the main system characteristics; and a simulator that enables detailed performance predictions (Sections 4.1 and 4.2).
- Third, we evaluate the impact on availability for three replica placement schemes. The results show that creating a new replica at the node that offers the fastest creation time can reduce unavailability by two orders of magnitude compared to a solution that aims for load balancing in terms of space, and by one order of magnitude compared to random placement (Section 4.3.5).
- Fourth, we demonstrate the feasibility of this approach through a prototype-based evaluation that confirms its high-throughput properties. The prototype implements a GridFTP server that employs the ThriftStore architecture. Additionally, we discuss the efficiency limitations of current GridFTP server designs, and present novel optimizations to address these limitations (Section 5).

2 Background and Related Work

This section presents background information on storage systems that use redundancy to improve data reliability (Sections 1 and 2) and discusses past work on analytical modeling of replicated storage systems (Section 3).

2.1 Redundant Storage Systems

A countless number of distributed storage systems use data redundancy (through replication or erasure coding) to improve reliability. These systems can be classified in three high-level categories based on their target deployment environment. First, *peer-to-peer storage systems* (e.g., OceanStore [3], TotalRecall [14], Farsite [15], and Ivy [16]) harness idle disk space from thousands of wide-area, distributed workstations, and use redundancy to reduce the probability of data loss and to increase availability. Second, *cluster-based storage systems* (e.g., PVFS [17], Ceph [2], and GoogleFS [1]) are hosted on dedicated, highly-available cluster nodes. Often the primary design concern in this case is achieving maximum throughput; hence, reliability is managed via simple replication techniques that do not compromise access performance. Third, closer to our system, *scavenged storage systems* (e.g., FreeLoader [18] and dCache [19]) aggregate idle storage space from LAN-connected, non-dedicated workstations. dCache, for example, combines heterogeneous disk-based storage systems (including donated storage) to offer a unified, high-performance data store. Similar to ThriftStore, dCache enables the inclusion of a tertiary storage system as a back-end reliable component, and uses file replication to improve reliability. dCache designers, however, do not provide guidance on how to provision the system to provide user-specified data reliability guarantees.

Finally, hierarchical storage management systems employ a similar architecture. Our system is different in that it integrates a set of volatile resources as the front-end, and that our study provides tools for adequate provisioning.

2.2 Design Techniques

The characteristics of the targeted deployment environment (e.g., the maintenance bandwidth available at the participating nodes as well as their failure and repair rates) are the main factors that drive the design of redundant storage systems. Blake and Rodrigues [20] argue that bandwidth constraints ultimately determine the achievable durable storage capacity regardless of the redundancy level used; Chun et al. [21] also reach the same conclusion. In these conditions, the following techniques have been used to reduce the volume of generated maintenance traffic:

Separating durability and availability. Chun et al. [21] observe that durability and availability can be decoupled to reduce maintenance costs in the presence of failures. Lefebvre and Feeley [22] provide a formula for determining the replica repair rate when durability is the only concern, and show that high availability requires much more frequent repairs.

ThriftStore, however, completely separates durability and availability management. Durability is entirely related to the characteristics of the durable component; while the desired level of availability is controlled by correctly provisioning system's resources (e.g., the bandwidth of the durable and volatile nodes as well as the replication level), hence allowing for a higher degree of flexibility to decide on the availability level without affecting durability.

Differentiating between transient and permanent failures can reduce repair traffic overhead. Carbonite [21] reintegrates replicas after transient failures; which implies that the system must track all replicas, including those located on offline nodes. We borrow this approach to reduce replica repair costs on the volatile nodes. Another approach, suggested by Blake and Rodrigues [20], is to use long timeouts when detecting failures. While this technique reduces the number of unnecessary repairs, it increases the time to repair a misclassified permanent failure, hence threatening durability.

Full replication and erasure coding. These are two widely employed data redundancy techniques. Full replication creates full copies of the data. It has the advantage of simplicity and low access overhead; however, it imposes higher repair traffic and storage overheads. Erasure coding, conversely, avoids whole object replication to reduce storage overhead; however, it imposes extra computational

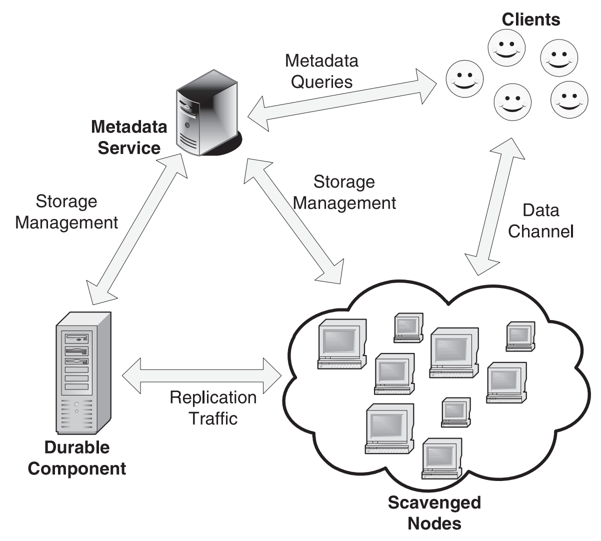


Fig. 1. ThriftStore architecture.

costs due to coding/decoding. ThriftStore is agnostic to the redundancy mechanism used as either erasure coding or full replication can be employed. As we design for high throughput, we have chosen to first explore a replication-based solution.

2.3 Analytical Models of Replicated Storage Systems

Markov chains have been used in the past to model replication-based systems. Ramabhardan and Pasquale [23] model data durability in replicated peer-to-peer storage systems. The model is analyzed to derive an expression for the expected object lifetime, to examine the impact of resource constrains (e.g., repair bandwidth limitations) on durability, and how to optimally tune these constrains to maximize durability.

Lian et al. [24] use Markov chains to model the impact of replica placement on durability in cluster-based storage systems. The model is used to derive an expression for mean time until the first permanent data loss occurs.

Our model is different in that it considers an asymmetric system. We base our model on the one proposed by Chun et al. [21]. We expand the model to add the durable component, and derive an expression for object availability.

3 System Model

This section describes the main components of ThriftStore (Section 1), details our assumptions (Section 2), and discusses a number of issues related to the system model (Section 3).

3.1 Architecturally Relevant Components

The architecture includes three components (Fig. 1):

3.1.1 A Large Number of Volatile Nodes

These nodes could be a subset of the desktops available in a company or a research institution. The volatile nodes maintain n copies of each object in the system. This forms a front-end storage area with substantial parallel IO channels, hence supporting high-performance data access.

3.1.2 One Durable Component

The durable component could be an external archival service or a complex set of devices (e.g., ATL) but, for the purpose of our analysis, we model it as a single durable component that maintains a copy of each object in the system. This forms a back-end data store that is able to recreate the data the volatile nodes may lose; hence, offering strong durability guarantees for the whole system.

3.1.3 A Logically Centralized Metadata Service

Clients locate objects by contacting a metadata service that maintains all metadata related to objects, detects replica failures, and makes replica placement decisions. Note that this service is just conceptually centralized as distributed implementations, which support better scalability and resilience to failures, are possible [2], [25] (albeit at the cost of increased complexity).

3.2 Definitions and Assumptions

3.2.1 Volatile Nodes' Failure Model: Transient and Permanent Failures

Permanent failures (e.g., a disk failure) cause data loss. Transient failures, on the other hand, do not cause data loss, but preclude instant access to data (e.g., a powered-off node). During transient failures, however, the system may waste bandwidth by triggering unnecessary repair actions. To reduce the cost of recovering after transient failures, replicas are reintegrated once they rejoin the system. Failures are detected using timeouts. Nodes declare their availability to the system via heartbeat messages sent to the metadata service.

3.2.2 Durability

It is the ability of the system to preserve data over time. All data objects have a copy on the durable component. Thus, the durability of the system is at least as good as that of the durable component. Since this is orders of magnitude better than the durability of the volatile nodes, for the practical goal of our evaluation we assume that the durable component never looses data (i.e., has 100 percent durability).

3.2.3 Availability

It is the ability to serve the data when requested. We assume that clients access the system only through the volatile nodes (Fig. 1). Thus, if a data object is not available on a volatile node (e.g., in case of a crash that hits all its replicas), the object is first copied from the durable node and, once it has at least a replica on a volatile node, the system makes it available to applications. *As a result, we define availability as the ratio of time a data object has at least one replica available on a volatile node*. The rationale behind this choice is twofold. First, we aim to avoid using the durable node to directly serve the application's read workload to minimize the load the durable node must serve (and to implicitly minimize its cost). Second, even if the durable node could directly serve client load, this flow will directly compete with other repair traffic. Thus, the available throughput for any single client will severely be impacted and applications may deem it unsatisfactory, making, in effect, the corresponding data object unavailable.

3.2.4 Replica Repair Model

The system seeks to maintain a minimum replication level n for all objects at all times on the volatile nodes. Once a failed volatile node is detected, the system reacts by creating additional copies to increase the replication level back to n.

Each volatile node limits the bandwidth allocated for replica repair to b. If the replication level on the volatile nodes goes down to 0 for a specific object, then the durable component creates, with a repair bandwidth limited to B, a new copy of the object on one of the volatile nodes. The new copy is then used to create additional replicas on other volatile nodes up to the replication level n.

3.3 Discussion

This section discusses a number of questions related to the system model.

1. *How does the system deal with write operations?* The proposed system mostly benefits a read-intensive workload. Writes can be handled in two ways that trade between throughput and durability. Pessimistic writes are executed first on the durable component then the normal system operation creates replicas on volatile nodes and makes the data available for reading. Alternatively, optimistic writes are first performed (with a predetermined redundancy) on the volatile nodes, then data is transferred in the background to the durable component. Also, workload characteristics or application-provided information can be exploited to improve write performance. For example, if an application only needs a fast "scratch" space for temporary data, the system can skip pushing the data to the back-end durable component.

   Regardless of the approach used to handle writes, our study demonstrates a reduced read load on the durable component. This, in turn, helps improve the write throughput by leaving a higher share of the durable component throughput available for the write load.
2. *Can workload and platform characteristics be exploited to further improve availability (or, equivalently, to reduce maintenance costs for a given availability level)?* Yes, it is possible to statically or dynamically tune the redundancy level of individual objects or the redundancy technique used to achieve a particular availability level while controlling the associated redundancy costs. On the one side, the system can exploit the workload characteristics in multiple ways. For example, the system can provide differentiated availability according to object popularity to increase perceived system availability while reducing the overall costs. Similarly, it is possible to prioritize the repair of frequently accessed objects to improve their availability. Finally, the system can exploit the fact that some redundancy techniques are better suited for some specific workloads than others: for example, full replication is well-suited for random access patterns and for small-sized files as it has high storage overhead, yet low runtime access latency (no processing required to access the object); erasure coding, however, has lower storage requirements compared to pure replication, yet increased data access latency (due to coding/encoding) thus it can better handle workloads with sequential access to large files. On the other side, the system can adapt to platform characteristics. For example, as suggested by Bhagwan et al. [14], the system can dynamically adjust the redundancy level using real-time estimates for the MTTF of the

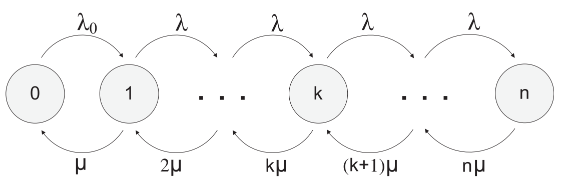


Fig. 2. Markov chain model.

volatile nodes. All these optimization techniques, however, are orthogonal to our study as we focus on evaluating the benefits and the tradeoffs brought by the basic properties of the hybrid architecture in the absence of any specific optimizations.

3. *Where do file system properties (e.g., security and consistency) fit in the proposed architecture?* We propose storage system architecture not a file system: high-level file system properties, such as security and consistency semantics, are outside the scope of our study. For example, the architecture does not impose the use of a specific access control solution or consistency model; these can be implemented on top of the existing architecture. However, the characteristics of the architecture (e.g., the fact that it is distributed) may impact the design of such features.

4 Availability Study

This section uses analytical modeling and simulations to study the effect of the following factors on data availability and on the associated maintenance costs:

- *the durable component characteristics:* the relevant characteristic is the durable component's read bandwidth,
- *the volatile nodes characteristics:* number of volatile nodes, their availability, and bandwidth,
- *the workload characteristics:* the number and size of the objects maintained by the system, and
- *the replication scheme characteristics:* replication level and replica placement strategy.

4.1 The Analytical Model

This section describes our analytical model (Section 1), derives an expression for object availability (Section 2), and discusses the model's accuracy and limitations (Section 3).

4.1.1 The Model

We model the number of replicas of an object that exist in the system as a birth-death process using a discrete-time, discrete-space Markov chain.

Fig. 2 illustrates the model. An object can have a maximum of n replicas. An object is in state k when k replicas exist on the volatile nodes. The object is in state 0 when it exists only on the durable node, a situation where we consider the object unavailable.

From a given state k, there is a transition to state $k+1$ with rate λ_k corresponding to the replica repair rate. For $k=0$, the repair rate λ_0 depends on the characteristics of the durable node and the size of the object. For $0<k<n$, the system is in repair mode and creates additional replicas (i.e., it "repairs" them) constantly. We assume the repair rate is capped at similar values on all participating nodes, thus $\lambda_k = \lambda$ for all $0<k<n$. This repair rate λ depends on the volatile nodes' repair bandwidth and the size of the data object.

Further, from a state $k>0$, transitions to the next lower state "$k-1$" happen with rate $k\mu$ as each of the k nodes holding a replica may fail with a failure rate μ which we assume to be the same for all volatile nodes.

4.1.2 An Expression for Object (Un)availability

We assume exponentially distributed repair and failure interarrival times. Using exponentials to model both the repair and failure rates produces a mathematically tractable model that can be analyzed analytically as an M/M/K/K queue (where K is, in fact, the replication level n). We discuss the validity of this assumption in the next section.

As a reminder, in this context, the object is unavailable while it is in state 0. The probability of being in state 0 (p_0) is given by [26]:

$$p_0 = \frac{1}{1+\sum_{k=1}^{n}\prod_{i=0}^{k-1}\frac{\lambda i}{\mu_{i+1}}}$$

However,

$$\lambda_k = \begin{cases} \lambda_0 & k=0, \\ \lambda & 0,<k\le n, \\ 0 & k>n, \end{cases}$$

and

$$\mu_k = k\mu \quad for\ k \ge 0.$$

As a result, after derivations, unavailability (p_0) becomes:

$$p_0 = \frac{1}{1+\sum_{k=1}^{n}\frac{\lambda_0}{\mu}\left(\frac{\lambda}{\mu}\right)^{k-1}\frac{1}{k!}}.$$

Two notations better reveal the meaning of the above formula:

- We refer to the term λ/μ as ρ; intuitively, ρ represents the *volatile nodes replica repair ratio*: the ratio between how fast the volatile nodes create new replicas and how fast they fail (or, equivalently, the number of replicas the system can create during the lifetime of a volatile node).
- Additionally, we refer to the term λ_0/μ as γ, which is *the durable node replica repair ratio*: the ratio between how fast the durable node creates new replicas (placed on the volatile nodes) and how fast these fail.

Using these notations:

$$p_0 = \frac{1}{1+\gamma\sum_{k=1}^{n}\frac{\rho^{k-1}}{k!}}.$$

A number of observations can be made: first, setting n much larger than ρ (i.e., significantly increasing the replication level) does not significantly increase availability (reduce p_0). The reason is that the term $k!$ is much larger than ρ^{k-1} for $n>\rho$. Second, if ρ is large enough, then a reasonable value of γ is enough to achieve good availability (i.e., low p_0).

Note that when $n=1$, availability depends only on the durable node repair ratio γ, which is intuitive. Also, as expected, there are decreasing incremental gains from increasing the replication level.

4.1.3 Model Accuracy and Limitations

A number of assumptions limit the accuracy of the analytical model, yet make it mathematically tractable and make it possible to derive a compact closed-form expression for availability that offers two primary benefits: first, it unveils the key relationships between the system characteristics, hence it is still useful for a coarse-grain, qualitative characterization of the system and, second, it offers a good approximation for availability at specific points in the parameters space, which enables validating the correctness of the simulator discussed in the next section.

The rest of this section summarizes the simplifying assumptions made by the analytical model.

First, *the model does not capture transient failures*. In other words, all failures (i.e., transitions from state k to state $k-1$) are assumed to be destructive. However, especially for a system built atop donated storage, nodes may depart and rejoin the system without actually losing data [27], [28].

Second, *the model assumes that failures are time-independent*. While this is an approximation frequently used, recent work by Schroeder et al. [28] analyzes failure traces from six large-scale production systems: high-performance computing sites and large data centers, and finds that disk failures are autocorrelated in time and have long-range time dependencies.

Third, *the model assumes exponentially distributed replica repair and lifetimes*. Empirical evidence supports the assumption that component lifetimes (e.g., time to permanent disk failure) are well-modeled by exponential distribution [23], [29], [30]; this assumption, however, is not well-supported for replica repair times.

Fourth, *the model analyzes the state of a single object*. In reality, several objects exist in the system and share the system's resources. This fact has a direct effect on the actual replica repair rate which varies depending on the number of objects that are concurrently repaired. The analytical model, however, is agnostic to the number of objects in the system, and assumes that the repair rate does not depend on the number of concurrent repairs. Another implication of this limitation is that the model does not capture the effect of replica placement decisions.

Fifth, *the model does not capture the fact that replicas of the same object can be repaired in parallel*. In reality, when an object is in replication level k, the missing $n-k$ replicas can be repaired in parallel. As a result, a more realistic model for λ_k will depend on the number of replica sources k and the number of replica destinations $n-k$, and it is expressed as $\lambda^{*}\min(n-k, k)$. However, to keep the analytical model tractable, λ_k is assumed to be constant irrespective of the current replication level of the object. This assumption is conservative in the sense that the analytical model uses a lower bound for the replica repair rate.

Finally, *the model has no time factor*. Our analysis is based on discrete-time, discrete-space Markov chain; hence, it predicts the availability of the system only in the equilibrium state (i.e., after running for long-enough time).

4.2 The Simulation Model

To overcome the limitations of the analytical model, we build a simulator that provides a more accurate view of the proposed system. The simulator is built using SimPy [31], a discrete-event simulation language. The rest of this section discusses the simulation model (Section 1), and its accuracy (Section 2).

4.2.1 The Model

Each volatile node in the simulator has limited bandwidth and disk space, while the durable node has limited bandwidth and unlimited storage space.

The simulator is driven by: 1) a trace of failure events (transient and permanent) for the volatile nodes (the traces will be discussed below), and 2) the number and size of the objects to be maintained. It simulates the behavior of both the durable and volatile nodes, and it applies different replica placement and repair polices. The simulator monitors the state of the objects (i.e., their replication level), and the amount of traffic generated by all nodes throughout the simulation.

The replica repair policy is simulated as follows: when the metadata service detects a volatile node failure, it attempts to increase the replication level of all objects maintained by the failed nodes back to the minimum replication level n. For each lost replica, the metadata service sends a repair request to a corresponding live replica hosted on another volatile node. If no live replica is available, the request is sent to the durable node. To model contention on access links and storage devices, repair requests are queued and served sequentially by the storage nodes (both durable and volatile). Hence, a node's upload channel is not time-shared and is dedicated for one transfer at a time. A repair request is processed as follows: The source node asks for a candidate destination node from the metadata service. The metadata service, in its turn, replies with a candidate destination based on the replica placement scheme employed. Once obtained, the source node informs the destination of the upcoming object transfer, and waits for acknowledgment to start the transfer. At the destination node, incoming transfer requests are queued and served in FIFO order; consequently, similar to the upload channel, a node's download channel is not time-shared as it is dedicated for one transfer at a time.

4.2.2 Model Accuracy

Assuming accurate traces, the simulation model addresses all the shortcomings of the analytical model discussed in Section 4.1.3) with one exception: network contention is modeled only on access links (i.e., the model assumes direct connectivity for all node pairs and no shared bottlenecks at the network core). Given that we target a LAN environment with good connectivity and high-speed core switches, this assumption is acceptable; further, our experiments in the evaluation section, which estimate the peak-aggregate-generated traffic, support the adequacy of this assumption.

4.3 Simulation-Based Evaluation

We use simulations to 1) gain additional insight beyond what the analytical model can offer, 2) compare the characteristics of our design with those of systems built atop of unreliable components only, and 3) evaluate the effect of the deployment platform and workload characteristics on data availability and on the generated maintenance traffic.

The simulator is driven by traces of node failure and repair events, thus the accuracy of the simulation results depends on the accuracy with which these traces reflect real-world events. We use Weibull distributions to generate transient failures' interarrival times. Heath et al. [32] argue that Weibull distributions model well desktop availability in an enterprise environment, Nurmi et al. [33] also reach the same conclusion. As for permanent failures, exponential distributions are widely accepted to model their interarrival time [21], [23], [29], [30], [34].

We use synthetic traces instead of real-life traces for two reasons. First, synthetic traces enable varying failure density, hence allowing for better investigation of key system trends. Second, to the best of our knowledge, there are no public desktop availability traces long enough to expose the impact of permanent disk failures where MTTF is in the order of years (the traces analyzed by other desktop availability studies are at most a few months long [32], [33], [35]). We have investigated the possible use of traces that characterize other environments (e.g., HPC clusters [36]). However, these traces capture only permanent failures, which is insufficient to drive the simulator.

Unless otherwise noted, simulations are configured as follows: the system maintains 32 TB of data divided into 1GB objects. The replication level n is set to 3, a common value used in distributed storage systems [1].

The impact of object size can be summarized as follows: On the one side, using small objects increases the opportunity for parallel repair; however, it produces large number of objects that generate a larger number of placement combinations; as a result, the simultaneous failure of any n nodes is likely to result in data loss. On the other side, using very large object size reduces the opportunity for parallel repair; however, it results in consuming fewer placement combinations, hence it is less likely that the failure of n nodes will affect the replica set of a specific object. A more detailed discussion on the impact of object size appears in [24].

The system contains 1,000 volatile nodes, each such node allocating up to $b = 2$ Mbps for replica maintenance. The durable node provides up to $B = 1$ Mbps read throughput. This value is within the range of the wide-area download throughput offered by Amazon S3 [37].

Further, each simulation generates its own four-year-long synthetic traces with the following parameters. For the transient failures, the Weibull distribution parameters are: 0.49 for the shape factor, and 10 days for the scale factor (these parameters are derived from [32], and the corresponding node MTTF is 20 days). For the permanent failures, the exponential distribution has a node MTTF of one year (similar to [21]).

All experiments report averages over at least 30 runs and 95 percent confidence interval. Hence, to reproduce the simulation results, one only needs to configure the random distribution functions with the above-mentioned parameters.

Finally, we report unavailability rather than availability as it is easier to plot on logarithmic scale.

4.3.1 Comparing the Analytical Model with Simulation

The first question we address is: *how close the analytical model is to simulations?* To answer this question, we estimate unavailability for various values of volatile nodes' bandwidth. Fig. 3 presents the results.

Two details of this experiment are worth discussing. First, since the analytical model does not model transient failures, in this simulation we only generate permanent failures. Second, to seed the analytical model, two parameters need to be estimated: the repair ratios ρ and γ. To do so, we need to estimate the creation rates λ and λ_0, and the failure rate μ implied in the generated traces. The replica failure rate μ depends on the volatile nodes' failure rate. Since the lifetime of a replica is equal to the lifetime of the node that maintains that replica, we can estimate μ as $1/MTTF_{vn}$, where $MTTF_{vn}$ is a volatile node's mean time to permanent failure. The replica creation rate λ depends on the amount of data that need to be repaired, and the volatile nodes' repair bandwidth. Let d be the average amount of data maintained by each volatile node, and b a volatile node's repair bandwidth. Consequently, one node's data can be recreated with a rate of b/d; therefore, on average, the creation rate λ of a single object can be estimated by $2 * b/d$.

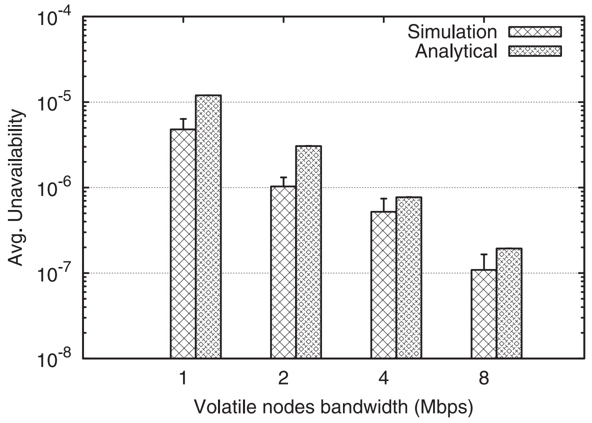


Fig. 3. Comparing the unavailability predicted by the analytical model and simulations (the lower the unavailability, the better).

In this experiment, there are 1,000 volatile nodes, and the average amount of data maintained by each node is $\mathrm{d} = 32 * 3/1,000\mathrm{TB} \approx 98$ GB; thus, for example, for the first data point, where $b = 1M$bps, $\lambda \approx 78$ replica copies per year. The same estimation also applies to λ_0 by replacing b with the durable node's repair bandwidth.

These estimates for the creation rates λ_0 and λ are conservative as they assume that a failed node's data are recreated only by one node (at b repair rate). In reality, data of a failed node is repaired in parallel by more than one volatile node. However, the degree of parallel repair is difficult to estimate, as it depends on the system state at the time of failure, which is related to how the objects to be repaired are distributed in the system, and on other repair operations processed at the same time competing for the available repair bandwidth. As a result, the analytical model we use here is seeded to make conservative availability predictions (i.e., higher unavailability) compared to simulations.

Nevertheless, the conservative prediction by the analytical model is still useful in that it defines a coarse lower bound for availability. Further, this expected result (the fact that the simulation predicts slightly better availability than the analytical model) increases our confidence with the accuracy of the simulator.

4.3.2 The Impact of the Stored Data Volume on Durability

To highlight the benefits of using a durable component, and the importance of adequate resource provisioning, this

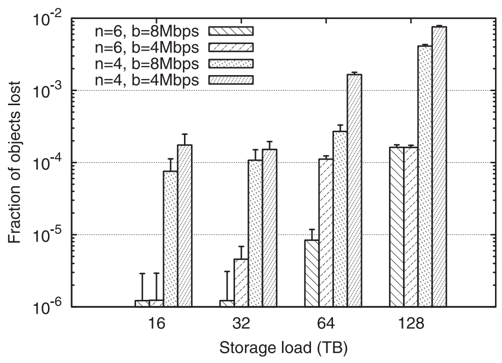


Fig. 4. The fraction of objects lost after four years. For storage load less than 16 TB, no objects were lost in any configuration. When aggressively increasing both the replication level and the bandwidth limits ($n = 8$ and $\mathrm{b} = 8$ Mbps), the system does not lose any objects for a storage load of up to and including 128 TB.

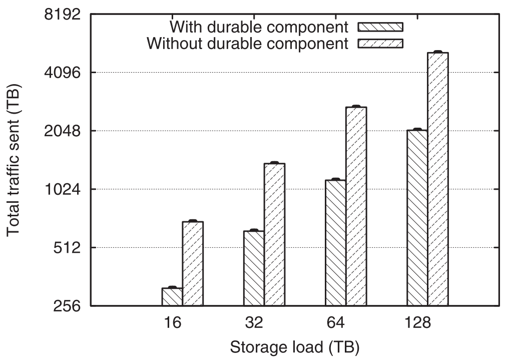
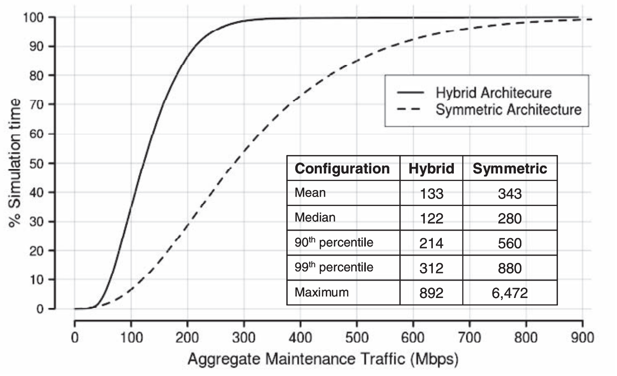


Fig. 5. Total traffic sent during the experiment (log scale). The system with durable component is configured with $B = 1$ Mbps, $b = 2$ Mbps, and $n = 3$. The system without durable component is configured with $b = 8$ Mbps, $n = 8$.



Configuration	Hybrid	Symmetric
Mean	133	343
Median	122	280
90th percentile	214	560
99th percentile	312	880
Maximum	892	6,472

Fig. 6. CDF of aggregate maintenance traffic generated for a storage load of 128 TB. For better visibility, the figure plots the curve for the asymmetric architecture up to the 99th percentile only.

section considers a system built atop unreliable, volatile nodes only. This system employs replication to provide both durability and availability. We are interested in answering the following question: *using replication on the volatile nodes only, what is the maximum data volume that can be preserved durably for a specific time interval given particular system characteristics (e.g., repair bandwidth and replication level)?*

To answer this question, we run simulations in which we remove the durable component (by setting its bandwidth to 0), and vary the storage load maintained by the system. Fig. 4 shows the fraction of objects lost at the end of the experiment.

As the storage load grows, each node maintains a higher data volume. Consequently, repairing a failed node datum requires more time, and this increases the likelihood of other, almost concurrent, failures that may cause data loss. To compensate for this effect and increase durability, the repair bandwidth and/or the replication level can be increased. For instance, using $n = 8$ and $b = 8$ Mbps, the system does not lose any objects while maintaining up to 128 TB. But, increased durability comes at the cost of increased maintenance traffic and disk space. Further, when the storage load increases beyond 128 TB, the system starts losing data again.

4.3.3 Reducing Durability Costs: Trading off Availability for Durability

Given the results of the previous experiment, we aim to quantify the benefits of adding a durable back-end storage component, even with a low access bandwidth. The questions we aim to answer are: *How much does the system save in terms of maintenance traffic and disk space when a durable component is added? Further, what is the impact on availability as a trade-off to increased durability when one replica is moved to a durable yet low-bandwidth component?*

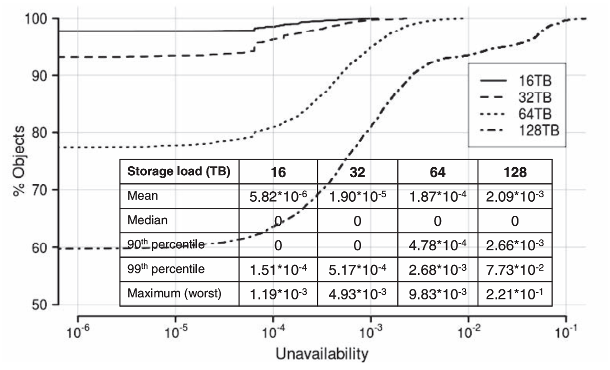
To address these questions, we run an experiment comparing the two systems. For the system with a durable component, we use our default configurations: $B = 1$ Mbps, $b = 2$ Mbps, $n = 3$ replicas. For the system without durable component, we use the minimum configuration in which the system was able to durably store all data up to 128 TB: $b = 8$ Mbps and $n = 8$.

Fig. 5 demonstrates that the system with the durable component generates 60 percent less maintenance traffic than the system without a durable component. For example, for a storage load of 128 TB, the system without durable component generates, on average, an aggregate traffic of about 343 Mbps, a significant burden even for a gigabit LAN compared to only 133 Mbps for the system with durable component.

Fig. 6 presents the cumulative distribution function (CDF), and the summary statistics, for the aggregate maintenance traffic generated over time for the previous experiment. The system with the durable component not only generates lower average maintenance traffic, but the generated traffic has lower variability, making the maintenance overhead more predictable and, as a result, reducing its impact on competing application traffic. Moreover, the peak bandwidth required by the system with durable component is seven times lower than the peak bandwidth required by the system without durable component, a major maintenance cost reduction.

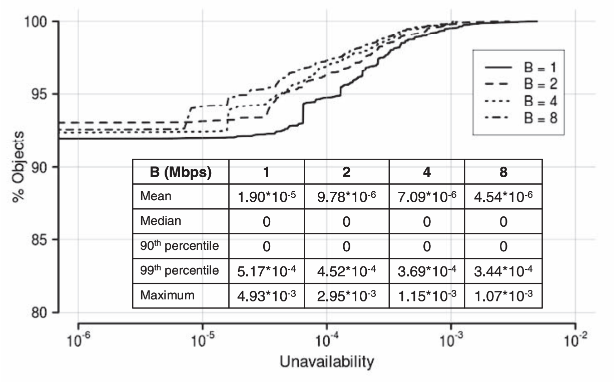
Finally, the system with a durable component requires significantly less disk space as it uses four replicas (one on the durable node and three on the volatile nodes) compared to eight replicas used by the system without a durable component.

The price paid for these savings is twofold: 1) a slightly more complex, asymmetric system that includes the durable component and, 2) higher unavailability (Fig. 7). Although unavailability increases as the storage load increases, the system is still able to provide acceptable average availability



Storage load (TB)	16	32	64	128
Mean	$5.82*10^{-6}$	$1.90*10^{-5}$	$1.87*10^{-4}$	$2.09*10^{-3}$
Median	0	0	0	0
90th percentile	0	0	$4.78*10^{-4}$	$2.66*10^{-3}$
99th percentile	$1.51*10^{-4}$	$5.17*10^{-4}$	$2.68*10^{-3}$	$7.73*10^{-2}$
Maximum (worst)	$1.19*10^{-3}$	$4.93*10^{-3}$	$9.83*10^{-3}$	$2.21*10^{-1}$

Fig. 7. CDF of unavailability (the lower the unavailability, the better) for the system with durable component while varying the volume of data stored. The system is configured with $B = 1$ Mbps, $b = 2$ Mbps, and $n = 3$. Note that the Y-axis starts at 50 percent.



B (Mbps)	1	2	4	8
Mean	$1.90*10^{-5}$	$9.78*10^{-6}$	$7.09*10^{-6}$	$4.54*10^{-6}$
Median	0	0	0	0
90th percentile	0	0	0	0
99th percentile	$5.17*10^{-4}$	$4.52*10^{-4}$	$3.69*10^{-4}$	$3.44*10^{-4}$
Maximum	$4.93*10^{-3}$	$2.95*10^{-3}$	$1.15*10^{-3}$	$1.07*10^{-3}$

Fig. 8. CDF of unavailability while varying the durable node's bandwidth (B). Note that the Y-axis starts at 80 percent.

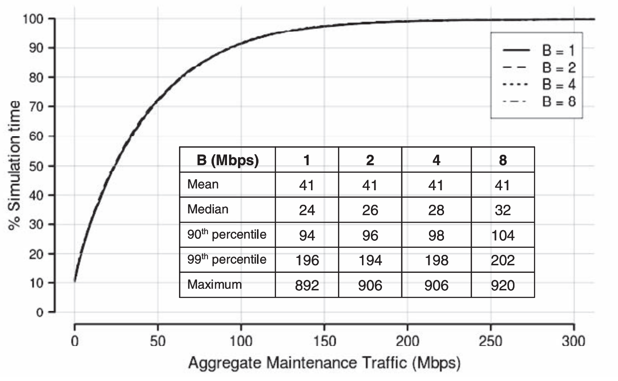
levels even for large storage volumes; moreover, a significant percentage of the objects (90 percent) have about the same availability level as the average. Further, a significant percentage of the objects have full availability (97 percent for the smallest storage load used in the 16 TB experiment, and 60 percent for the largest storage load used in 128 TB).

Finally, note that for the system without a durable component, durability and availability are, in effect, identical, and for the parameters we consider, the system offers full availability for storage load up to 128 TB. However, this is at the expense of extra storage and maintenance traffic.

4.3.4 Provisioning System Resources

To provide insight on system resource provisioning, we explore the effect of bandwidth limits at the durable and the volatile nodes, as well as the effect of the replication level, on the achieved availability and generated maintenance traffic.

The impact of bandwidth limitations at the durable node on availability and volume of generated traffic. Fig. 8 plots the CDF of object unavailability while increasing the durable node's bandwidth. Increasing the bandwidth of the durable component has little impact on availability. For example, in this experiment, reducing average unavailability by one order of magnitude requires increasing the durable component's bandwidth by more than eightfold while keeping the volatile node's bandwidth and the replication level constant.



B (Mbps)	1	2	4	8
Mean	41	41	41	41
Median	24	26	28	32
90th percentile	94	96	98	104
99th percentile	196	194	198	202
Maximum	892	906	906	920

Fig. 9. CDF of aggregate maintenance traffic generated while varying the durable node's bandwidth (B).

This is expected as the role of the durable component is limited to recreating only the first replica of a lost object. After this, only the volatile nodes' repair bandwidth influences how fast additional replicas are created before losing the object again (more on this in Section 5).

Increasing the stable component bandwidth does not impact the generated maintenance traffic (Fig. 9). The slightly better availability is achieved at the cost of enabling faster recovery directly from the durable node. This is an observation with direct practical impact: for example, when considering outsourcing the durable component to storage utility that offers two levels of service differentiated by their access performance, the system deplorers should not pay for the "premium" service as it only marginally impacts availability.

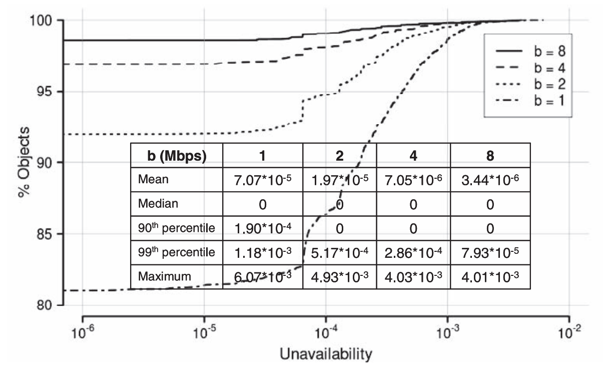
The impact of bandwidth limitations at the volatile nodes on availability and volume of generated traffic. Unlike the effect of the durable node's bandwidth, the volatile node's bandwidth has a marked impact on availability (Fig. 10). For example, reducing average unavailability by one order of magnitude requires increasing the volatile nodes' bandwidth fourfold while keeping the durable component's bandwidth and the replication level constant.

Note that this improvement is reflected noticeably on the distribution: the 90th and 99th percentile decrease along with the average; however, the maximum unavailability still does not improve proportionally.

Interestingly, the volume of maintenance traffic generated does not increase proportionally with the significant observed improvement in availability. The improvement in availability is achieved at the cost of allowing larger traffic spikes, which help repair after failure bursts. This is demonstrated by the increase in the length of the right tail of the traffic distribution presented in Fig. 11.

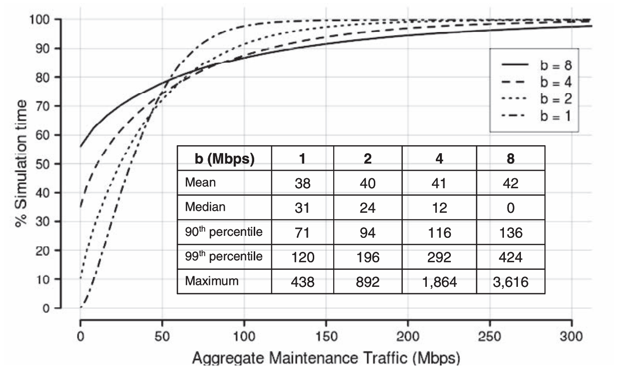
The impact of the replication level. As expected, increasing the replication level decreases unavailability (Fig. 12). The quantitative result is worth mentioning: for each additional replica, average unavailability decreases by one order of magnitude. We assume that the interconnection has appropriate capacity to support the desired replication levels.

Note that for all replication levels, at least 92 percent of the objects are fully available; further, and equally



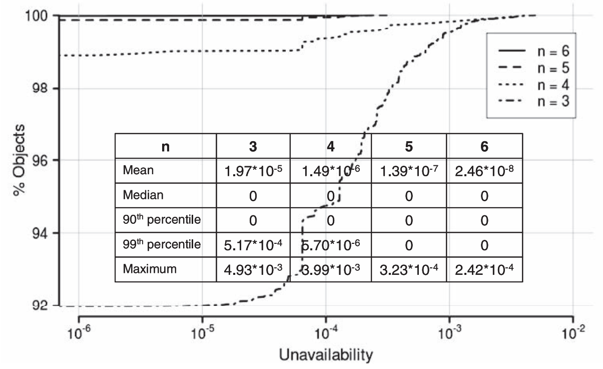
b (Mbps)	1	2	4	8
Mean	$7.07*10^{-5}$	$1.97*10^{-5}$	$7.05*10^{-6}$	$3.44*10^{-6}$
Median	0	0	0	0
90th percentile	$1.90*10^{-4}$	0	0	0
99th percentile	$1.18*10^{-3}$	$5.17*10^{-4}$	$2.86*10^{-4}$	$7.93*10^{-5}$
Maximum	$6.07*10^{-3}$	$4.93*10^{-3}$	$4.03*10^{-3}$	$4.01*10^{-3}$

Fig. 10. CDF of unavailability while varying the volatile nodes' bandwidth (b). Note that the Y-axis starts at 80 percent.



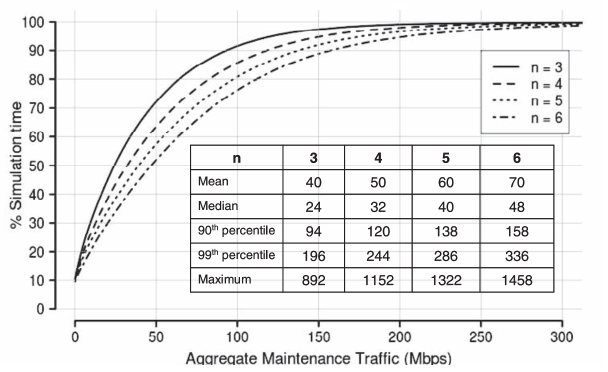
b (Mbps)	1	2	4	8
Mean	38	40	41	42
Median	31	24	12	0
90th percentile	71	94	116	136
99th percentile	120	196	292	424
Maximum	438	892	1,864	3,616

Fig. 11. CDF of total maintenance traffic generated while varying the volatile nodes' bandwidth (b).



n	3	4	5	6
Mean	$1.97*10^{-5}$	$1.49*10^{-6}$	$1.39*10^{-7}$	$2.46*10^{-8}$
Median	0	0	0	0
90th percentile	0	0	0	0
99th percentile	$5.17*10^{-4}$	$5.70*10^{-6}$	0	0
Maximum	$4.93*10^{-3}$	$3.99*10^{-3}$	$3.23*10^{-4}$	$2.42*10^{-4}$

Fig. 12. CDF of object unavailability while varying the replication level n.



n	3	4	5	6
Mean	40	50	60	70
Median	24	32	40	48
90th percentile	94	120	138	158
99th percentile	196	244	286	336
Maximum	892	1152	1322	1458

Fig. 13. CDF of total aggregate replication bandwidth consumed while varying the replication level (n).

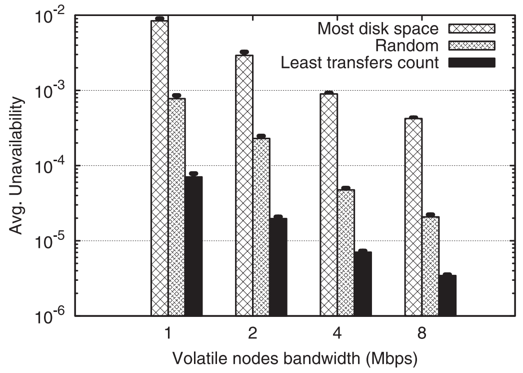


Fig. 14. Comparing average unavailability for three different replica placement schemes.

important, the maximum unavailability is reduced proportionally as well.

Fig. 13 presents the corresponding maintenance traffic distribution. The significant availability gain achieved when increasing the replication level comes at the cost of increasing the storage space used and maintenance traffic generated: increasing the replication level from three to six results in 75 percent increase in the mean, 71 percent increase in the 99th percentile, and 63 percent increase in the peak traffic.

4.3.5 The Impact of the Replica Placement Scheme

The replica placement scheme decides where to recreate a failed replica. For all the experiments presented so far, the system was configured to choose a destination node with the least number of pending transfers so that lost replicas can be recreated as quickly as possible; we call this scheme "least transfers count placement." Implementing this scheme, however, requires global information on system's state that may be costly to obtain. Thus, the final question we address in this section is: *What is the impact of the replica placement decisions on data availability?*

To answer this question, we compare two replica placement schemes and the one presented above. First, with random placement, the destination node is chosen randomly. Second, with most disk space placement, the replica is placed at the node with the maximum available disk space. The advantage of random placement is that it does not require global information; while the advantage of placement based on disk space is that it consistently balances the data volumes stored across all nodes.

Fig. 14 compares the three placement schemes. Surprisingly, the "most disk space" scheme performs much worse than the other two schemes, while the "least transfers count" scheme achieves much better availability. The reason is that the latter enables better utilization of the repair bandwidth across the volatile nodes by distributing the replica repair work evenly across available access links. Finally, we note that the availability loss caused by the random placement solution (roughly a loss of one "nine") is the price to pay for a solution that does not use global information.

5 Use Case: A GridFTP Server

While the previous section focused on exploring the relationships between data availability and various system characteristics (e.g., replication level, available repair bandwidth, and replica placement solution), this section

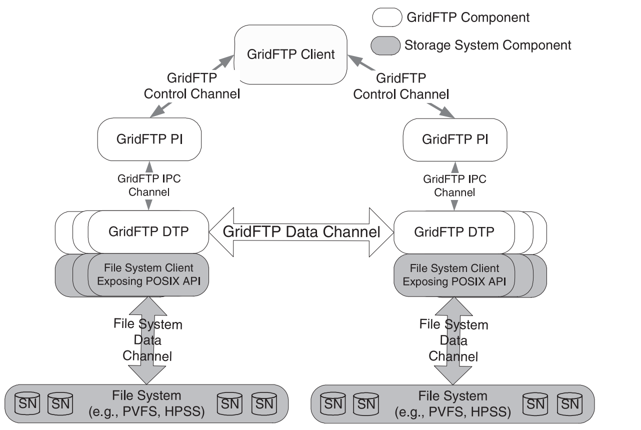


Fig. 15. Globus's GridFTP server in a typical deployment in the context of a third-party transfer. A server consists of a single front-end protocol interpreter (PI) node that parses clients' commands, and a set of nodes that run data transfer processes (DTP) which handle requests from the PI and are responsible for accessing and transferring stored data.

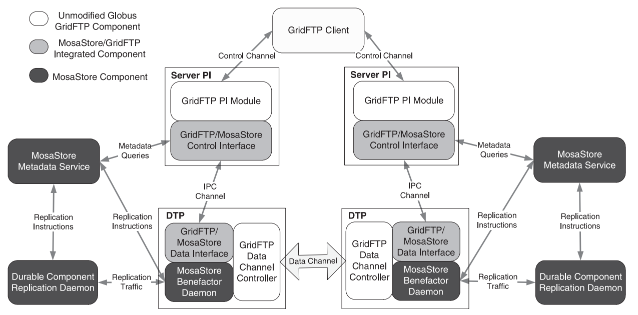

Unmodified Globus GridFTP Component
MosaStore/GridFTP Integrated Component
MosaStore Component
GridFTP Client
Control Channel
Control Channel
Server PI
GridFTP PI Module
GridFTP/MosaStore Control Interface
MosaStore Metadata Service
Metadata Queries
IPC Channel
Replication Instructions
DTP
GridFTP/ MosaStore Data Interface
MosaStore Benefactor Daemon
GridFTP Data Channel Controller
Data Channel
Durable Component Replication Daemon
Replication Traffic

Fig. 16. GridFTP/MosaStore integrated architecture.

aims to explore the practical feasibility of this architecture, and to demonstrate its high I/O throughput characteristics.

To this end, we use the GridFTP server usage scenario mentioned in Section 1. GridFTP [6] has become the data access protocol of choice for data-intensive scientific communities. The protocol includes new features (e.g., striping and parallel transfers) that enable harnessing the independent I/O paths potentially offered by the deployment environment. More relevant to our work, GridFTP deployments are customarily supported by high-end hardware resources that offer high I/O access rates. We aim to demonstrate that our approach can reduce the cost of GridFTP deployments while retaining their high throughput.

We build a GridFTP server based on the proposed hybrid architecture. Our prototype integrates components from two systems: MosaStore, a scavenged storage system [13], and the Globus' project GridFTP server [6]. The main challenges in this integrated system relate to transparency (i.e., users should not perceive any difference between the service provided by the new GridFTP server and a standard GridFTP server) and efficient use of storage resources to provide high throughput.

The rest of this section discusses limitations of current GridFTP servers (Section 1), describes in detail our GridFTP server design (Section 2), and presents an evaluation of the prototype (Section 3).

5.1 Current GridFTP Servers' Limitations

This section discusses limitations of current GridFTP deployments or design.

5.1.1 High Deployment Costs

To enable high I/O access rates, current GridFTP deployments are typically supported by high-end hardware resources such as parallel file systems (e.g., PVFS [17] and HPSS [38]) over dedicated clusters (Fig. 15). The cost of these resources raises the barrier for research groups to join, collaborate, and share data in a grid.

Our hybrid storage architecture reduces the cost of GridFTP deployments: a low-cost Automated Tape Library (ATL) can be used as the durable component, while the volatile storage is obtained by harnessing the free, underused storage capacity of desktops that exist in any research organization, and are already maintained by site administrators. On the performance and scalability side, scavenging idle storage has two advantages: first, desktops at the same site are usually connected via high-bandwidth LANs; second, as the site grows, more desktops participate in the system, inherently increasing the I/O bandwidth and the storage space of the system. However, a new GridFTP design is required to enable the use of these resources.

5.1.2 Limited Efficiency

Existing GridFTP server designs do not exploit data locality. This is a result of accessing a parallel file system that uses striping (thus places different parts of a file on different storage nodes) through a POSIX API that cannot expose the location of these stripes. The lack of location information precludes the ability to allocate the task of transferring the data corresponding to a file stripe precisely to the node that persistently stores the stripe thus leading to additional data transfer overheads.

5.2 A Scavenged GridFTP Server

This section presents the design of a GridFTP server based on our proposed hybrid architecture, which addresses the above-mentioned limitations: it scavenges storage space to reduce deployment cost, and it keeps track and exposes stripe location information to enable high performance.

5.2.1 Server Design and Components

As mentioned, our GridFTP server design integrates components from two systems (Fig. 16):

First, *Globus' GridFTP* server consists of a single front-end protocol interpreter (PI) node that parses GridFTP clients' commands, and a set of nodes that run data transfer processes (DTP) which handle requests from the PI and are responsible for transferring stored data (Fig. 15). The *GridFTP framework* [12] is a modular framework designed to facilitate building GridFTP servers on top of various storage systems. Building a GridFTP server using this framework requires the implementation of two interfaces to access the underlying storage system: the *control interface* which is integrated with the PI module, and the *data interface* which is integrated with the DTPs and handles access to stored data and its transfer.

Second, *MosaStore* [13] is a highly configurable storage system designed to harness unused storage space from LAN-connected workstations. MosaStore's design consists of three components: a metadata service, a set of "benefactor" nodes that donate storage space, and the clients that access the system. Similar to the standard practice in parallel file system design, MosaStore employs striping to increase performance: files are divided into chunks distributed among the benefactors. MosaStore's metadata service maintains information related to available space, system's namespace, file attributes, and mappings from files to chunks and from chunks to benefactors. The benefactor nodes use soft-state registration to declare their status (on-/offline, and available disk space) to the management service, and serve client requests to store/retrieve data chunks. In a nutshell, a client performs a read operation by asking the metadata service for the file's chunks-benefactors mapping. Once obtained, the client pulls the data directly from the benefactors.

Using components from these two systems, we build a new GridFTP server (Fig. 16). MosaStore's metadata service takes over the role of the metadata service in the hybrid architecture, while the benefactors represent the volatile nodes; moreover, the benefactors also run the GridFTP DTPs to handle data transfers. Finally, the newly integrated server includes an additional component for the durable node.

Fig. 16 presents the integrated architecture and highlights the newly added components:

- *The Server PI* parses GridFTP commands sent by the client, and invokes the GridFTP/MosaStore control interface to handle the requests. Note that the interface between the PI and the GridFTP client (i.e., the control channel protocol) did not change, hence a standard GridFTP client can be used to access the server.
- *The DTPs* run on the donated storage nodes (representing the volatile nodes in the hybrid architecture). Each DTP is composed of three components: *1)* MosaStore's benefactor daemon which manages local storage and replication requests from the metadata service, *2)* the data channel controller implements the standard GridFTP data channel protocol and handles data movement from/to the node, and *3)* the data interface which handles data access requests from the server PI by conveying data between the benefactor daemon and the data channel controller.
- *MosaStore's Metadata Service* resembles the centralized metadata service in our proposed hybrid architecture: it maintains the system's metadata, detects failed nodes, and makes replica placement decisions.
- *The Durable Component Daemon* operates on the durable component where it handles replication requests. Note that the durable component does not run a DTP as it has no role in serving data to clients.

5.2.2 Example of Component Interaction

We use a third-party transfer to present the interaction among these components. Briefly, a third-party transfer is a direct data transfer between two GridFTP servers arbitrated by a GridFTP client. Specifically, it works as follows:

- *Identifying source DTPs.* The GridFTP client sends a read command in passive mode to the source server. The PI at the source server parses the command and invokes the GridFTP/MosaStore control interface to handle it. The control interface, in turn, contacts the metadata service asking for the segment-to-benefactor mapping. Once obtained, the control interface replies to the PI with a set of IP/Port addresses of the DTPs responsible for the file. The PI passes the addresses back to the client as a response to the passive read command.
- *Relaying the source DTPs to the destination server.* The client sends a write command in active mode to the destination server. As part of the active write parameters, the client passes the set of source DTP addresses (obtained previously from the source server) to the destination server.
- *Identifying the destination DTPs.* The PI at the destination server invokes the control interface passing it the source DTPs. The control interface, consequently, contacts the metadata service asking for a set of benefactors that may provide the required storage space.
- *Data movement.* At this point, the control interface at the destination server has identified the source and destination DTPs. Subsequently, the control interface delegates the write command to the DTPs running on the destination benefactors which pull the data directly and in parallel from the DTPs running on the benefactors of the source server.
- *Replication management.* Once data are written to the destination benefactors, the background replication daemon, called by the metadata service, creates the required number of replicas on other benefactors and on the durable node.

5.3 Evaluation

We evaluate our prototype on a testbed of 22 nodes each with a 2.33 GHz quad-core CPU, 4 GB memory, and 1 Gbps NICs. We compare our server's performance with the performance of unmodified GridFTP servers running over NFS [39] and PVFS [17]. It is important to note that PVFS is optimized for high-throughput operations, and assumes dedicated storage nodes in its design; consequently, PVFS does not handle storage nodes' failures.

To assess the performance of our design under high load, we set up a GridFTP server supported by 10 storage nodes, and accessed by 40 clients running on 10 other machines. Clients are started at 5 s intervals and each client reads 100 files of 100 MB each. Both MosaStore and PVFS were configured with 1 MB block size and stripe size of 4.

Fig. 17 shows the throughput aggregated over all clients. We have used the default configuration for the NFS setup, which was able to saturate the NFS server network access link (1 Gbps), hence no other configuration can achieve a better performance. With regard to PVFS, we have explored a large number of different configurations (e.g., strip size and group strip factor), all of which resulted in more or less the same performance.

This experiment illustrates the ability of our system to efficiently harness the available I/O channels and to scale to support an intense workload. Further, GridFTP/Mosastore achieves 60 percent increase in aggregate throughput compared to out-of-the-box GridFTP/PVFS setup. This performance increase is mainly due to GridFTP/Mosastore integrated server's ability to exploit data locality (which we discussed in detail in Section 5.1).

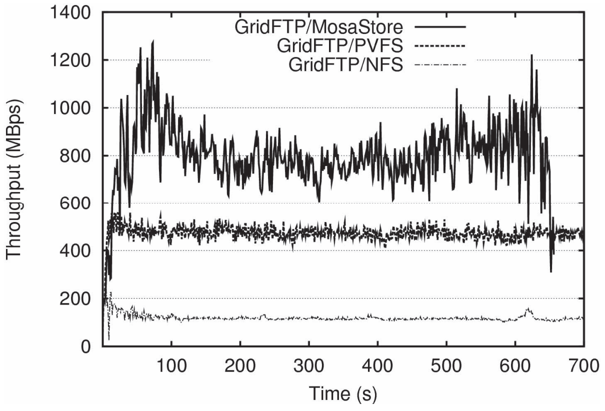


Fig. 17. Aggregate throughput for 40 clients reading 100 files of 100 MB each. The GridFTP server is supported by 10 storage nodes each connected at 1 Gbps.

Finally, the peaks at the beginning and the end of the experiment are primarily the result of lower contention for shared disk resources. Since the experiment was set up such that clients start 5 s apart, at the beginning and at the end of the experiment, there were fewer clients than in the middle of the experiment. As a result, at the beginning and at the end of the experiment, there is lower contention on disks (as they are able to deliver maximal throughput for one sequential read operation). A second effect that plays here as well is related to our (simplified) way of estimating the aggregate achieved I/O throughput: we use averages computed for each file transfer rather than finer granularity estimates. In more details: to estimate the aggregate delivered throughput, we compute the observed throughput for each download operation (as transfer size divided by duration), then, for each time interval, we sum the throughputs for all active I/O operations. Since at the beginning and at the end of the experiment there are fewer clients than in the middle of the experiment, the average throughput for these first (and last) file operations is higher and distributed by our simplified method to compute aggregated over the entire life of the file read operation.

6 Conclusions

This study demonstrates the feasibility of a low-cost storage architecture, named ThriftStore, which offers the durability and access performance characteristics of a well-endowed system. ThriftStore is based on integrating large number of volatile components and low-bandwidth durable components. This hybrid architecture allows for complete separation of the two components of data reliability—*durability* and *availability*. This decoupling, in turn, creates an opportunity to trade-off availability—the less critical property—for lower replica repair overhead, while maintaining strong durability guarantees.

We presented analytical- and simulation-based tools that are essential to analyze and provision this class of hybrid storage systems. We used these tools to investigate the availability-repair overhead trade-offs enabled by ThriftStore. Our results are summarized as follows:

- Adding a back-end durable component brings a number of gains. First, strong durability for the whole system becomes resilient to fatal concurrent failures (i.e., failures that hit all replicas of a set of objects), the main threat to durability in replicated storage systems. Second, a dramatic reduction in replica repair costs and the variability of the generated repair traffic, while offering the same durability guarantees. Particularly, the presence of a durable component halves the generated replica repair traffic and the used storage space, and reduces the required peak repair bandwidth by almost one order of magnitude. These advantages, however, come at the cost of the additional complexity of an asymmetric data replication scheme and slightly lower availability.
- Increasing the durable component's bandwidth has limited impact on improving availability; however, increasing the volatile nodes' bandwidth improves average availability at the cost of allowing higher peak repair traffic. Further, increasing the replication level improves both the average and minimum availability at the cost of higher peak and total repair traffic and used storage space.
- A replica placement scheme that creates a new replica at the node that offers the fastest creation time can reduce unavailability by two orders of magnitude compared to a solution that aims for load balancing in terms of space, and by one order of magnitude compared to random placement.

Finally, we present a prototype use-case application: a GridFTP server that employs the proposed hybrid architecture, which demonstrates that our approach can reduce the cost of data access infrastructures while maintaining their high-performance characteristics. Additionally, we improve the design of GridFTP server architecture to exploit data locality and reduce the data transfer overheads. The evaluation of our new GridFTP prototype proves the high-throughput potential of the system and its ability to scale under heavy workload.

A trade-off worth investigating, which we leave as future work, is the availability-energy trade-off. In particular, a number of disk energy conservation techniques are based on switching disks to low-power mode after a specific idle period [40]. Such techniques reduce energy consumption at the cost of lower availability as they increase replica repair times and client service time, especially in the case of load surges. The simulator we have developed can be modified to include energy as another managed resource hence enabling the exploration of energy-related techniques in the context of our architecture, and the study of their relationship with other system factors, such as the replica placement scheme.

References

[1] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," *Proc. Symp. Operating Systems Principles*, 2003.
[2] S.A. Weil, S.A. Brandt, E.L. Miller, D.D.E. Long, and C. Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," *Proc. USENIX Symp. Operating Systems Design and Implementation*, 2006.
[3] J. Kubiatowicz et al., "OceanStore: An Architecture for Global-Scale Persistent Storage," *ACM SIGARCH Computer Architecture News*, vol. 28, pp. 190-201, 2000.
[4] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-Area Cooperative Storage with CFS," *ACM SIGOPS Operating Systems Rev.*, vol. 35, pp. 202-215, 2001.

[5] C. Stone, "Cost of Ownership of Disk-Based vs. Digital Data Tape-Based Video Archives," 2008.

[6] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke, "GridFTP: Protocol Extensions to FTP for the Grid," Global Grid Forum GFD-RP, vol. 20, 2003.

[7] Amazon Web Services, http:/s3.amazonaws.com, 2010.

[8] Flickr,http://www.flickr.com, 2009.

[9] I. Raicu, I.T. Foster, and Y. Zhao, "Many-Task Computing for Grids and Supercomputers," *Proc. Workshop Many-Task Computing on Grids and Supercomputers (MTAGS '08),* pp. 1-11, 2008.

[10] Z. Zhang, A. Espinosa, K. Iskra, I. Raicu, I. Foster, and M. Wilde, "Design and Evaluation of a Collective I/O Model for Loosely-Coupled Petascale Programming," *Proc. Workshop Many-Task Computing on Grids and Supercomputers (MTAGS '08),* 2008.

[11] I. Raicu, I. Foster, Y. Zhao, A. Szalay, P. Little, C. Moretti, A. Chaudhary, and D. Thain, "Towards Data Intensive Many-Task Computing," *Data Intensive Distributed Computing: Challenges and Solutions for Large-Scale Information Management,* IGI Global Publishers, 2009.

[12] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, "The Globus Striped GridFTP Framework and Server," *Proc. ACM/IEEE Conf. Supercomputing,* 2005.

[13] S. Al-Kiswany, A. Gharaibeh, and M. Ripeanu, "The Case for a Versatile Storage System," *Proc. Workshop Hot Topics in Storage and File Systems,* 2009.

[14] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G.M. Voelker, "Total Recall: System Support for Automated Availability Management," *Proc. Symp. Networked Systems Design and Implementation,* 2004.

[15] A. Adya, W.J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J.R. Douceur, J. Howell, J.R. Lorch, M. Theimer, and R.P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," *Operating Systems Rev.,* vol. 36, pp. 1-14, 2002.

[16] A. Muthitacharoen, R. Morris, T.M. Gil, and B. Chen, "Ivy: A Read/Write Peer-to-Peer File System," *Proc. Symp. Operating Systems Design and Implementation,* 2002.

[17] P.H. Carns, W.B. Ligon, III, R.B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," *Proc. Ann. Linux Showcase and Conf.,* 2000.

[18] S.S. Vazhkudai, X. Ma, V.W. Freeh, J.W. Strickland, N. Tammineedi, and S.L. Scott, "FreeLoader: Scavenging Desktop Storage Resources for Scientific Data," *Proc. ACM/IEEE Conf. Supercomputing,* 2005.

[19] P. Fuhrmann, "dCache, the Commodity Cache," *Proc. Conf. Mass Storage Systems and Technologies,* 2004.

[20] C. Blake and R. Rodrigues, "High Availability, Scalable Storage, Dynamic Peer Networks: Pick Two," *Proc. Workshop Hot Topics in Operating Systems,* 2003.

[21] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M.F. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient Replica Maintenance for Distributed Storage Systems," *Proc. Conf. Networked Systems Design and Implementation,* 2006.

[22] G. Lefebvre and M.J. Feeley, "Separating Durability and Availability in Self-Managed Storage," *Proc. ACM SIGOPS European Workshop,* 2004.

[23] S. Ramabhadran and J. Pasquale, "Analysis of Long-Running Replicated Systems," *Proc. Int'l Conf. Computer Comm.,* 2006.

[24] Q. Lian, W. Chen, and Z. Zhang, "On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems," *Proc. Int'l Conf. Distributed Computing Systems,* 2005.

[25] S. Brandt, E. Miller, D. Long, and L. Xue, "Efficient Metadata Management in Large Distributed Storage Systems," *Proc. Conf. Mass Storage Systems and Technologies,* 2003.

[26] L. Kleinrock, *Queueing Systems. Volume 1: Theory.* Wiley-Interscience, 1975.

[27] B. Schroeder and G. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," *Proc. Conf. Dependable Systems and Networks,* 2006.

[28] B. Schroeder and G.A. Gibson, "Understanding Disk Failure Rates: What Does an MTTF of 1,000,000 Hours Mean to You?" *ACM Trans. Storage,* vol. 3, p. 8, 2007.

[29] F. Dabek, "A Distributed Hash Table," PhD dissertation, Massachusetts Inst. of Technology, 2005.

[30] M. Rausand and A. Hoyland, *System Reliability Theory: Models, Statistical Methods, and Applications.* Wiley-Interscience, 2004.

[31] Simpy, http://simpy.sourceforge.net, 2009.

[32] T. Heath, R. Martin, and T.D. Nguyen, "The Shape of Failure," *Proc. Workshop Evaluating and Architecting System Dependability,* 2001.

[33] D. Nurmi, J. Brevik, and R. Wolski, "Modeling Machine Availability in Enterprise and Wide-Area Distributed Computing Environments," *Lecture Notes in Computer Science,* pp. 432-441, Springer, 2005.

[34] H. Weatherspoon, "Design and Evaluation of Distributed Wide-Area On-Line Archival Storage Systems," PhD dissertation, Univ. of California Berkeley, 2006.

[35] W.J. Bolosky, J.R. Douceur, D. Ely, and M. Theimer, "Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs," *Proc. ACM SIGMETRICS,* 2000.

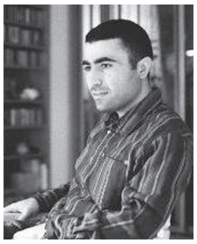
[36] B. Schroeder and G. Gibson, "The Computer Failure Data Repository (CFDR): Collecting, Sharing and Analyzing Failure Data," *Proc. ACM/IEEE Conf. Supercomputing,* 2006.

[37] M.R. Palankar, A. Iamnitchi, M. Ripeanu, and S. Garfinkel, "Amazon S3 for Science Grids: A Viable Solution?" *Proc. Workshop Data-Aware Distributed Computing,* 2008.

[38] R. Coyne, H. Hulen, and R. Watson, "The High Performance Storage System," *Proc. ACM/IEEE Conf. Supercomputing,* 1993.

[39] Sun Microsystems et al., "NFS: Network File System Protocol Specification," *RFC 1094,* 1988.

[40] E. Pinheiro, R. Bianchini, and C. Dubnicki, "Exploiting Redundancy to Conserve Energy in Storage Systems," *ACM SIGMETRICS Performance Evaluation Rev.,* vol. 34, p. 26, 2006.



Abdullah Gharaibeh received the MASc degree in computer engineering from the University of British Columbia. He is currently working toward the PhD degree in the Computer Engineering Department at the University of British Columbia. His interests are in the design and evaluation of high-performance distributed systems and GPU-based computing.

Samer Al-Kiswany received the BSc degree in 2003 from the Jordan University of Science and Technology, and the MSc degree in 2007 from the University of British Columbia (UBC), where he is currently working toward the PhD degree in the Electrical and Computer Engineering Department. His research interests are in distributed systems with special focus on high-performance computing systems. He is a student member of the IEEE.

Matei Ripeanu received the PhD degree in computer science from The University of Chicago. He is currently an assistant professor with the Computer Engineering Department of the University of British Columbia. He is broadly interested in large-scale distributed systems: peer-to-peer and Grid systems.