Exploring Data Reliability Tradeoffs in Replicated Storage Systems

Abdullah Gharaibeh, Matei Ripeanu
Electrical and Computer Engineering Department, University of British Columbia
{abdullah, matei}@ece.ubc.ca

ABSTRACT

This paper explores the feasibility of a cost-efficient storage architecture that offers the reliability and access performance characteristics of a high-end system. This architecture exploits two opportunities: First, scavenging idle storage from LAN-connected desktops not only offers a low-cost storage space, but also high I/O throughput by aggregating the I/O channels of the participating nodes. Second, the two components of data reliability – durability and availability – can be decoupled to control overall system cost. To capitalize on these opportunities, we integrate two types of components: volatile, scavenged storage and dedicated, yet low-bandwidth durable storage. On the one hand, the durable storage forms a low-cost 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.

While integrating these components has the potential to offer a unique combination of high throughput, low cost, and durability, 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., replica placement scheme) on data availability and the associated system costs (e.g., maintenance traffic). Further, we implement and evaluate a prototype of the proposed architecture: namely a GridFTP server that aggregates volatile resources. Our evaluation demonstrates an impressive, up to 800MBps transfer throughput for the new GridFTP service.

Categories and Subject Descriptors

D.4.3 [**Operating Systems**]: File Systems Management – *Distributed File Systems.* D.4.8 [**Operating Systems**]: Performance – *Measurements, Modeling and Prediction.*

General Terms

Reliability, Measurement, Performance.

Keywords

Storage System, Reliability, GridFTP, MosaStore.

1. INTRODUCTION

As the volume of digital data grows, reliable, low-cost storage systems that do not compromise on access performance are increasingly important. A number of storage systems (e.g., tape libraries, optical jukeboxes) provide high reliability coupled with low I/O throughput. However, as throughput requirements grow, using high-end components leads to increasingly costly systems. A number of existing distributed storage systems (e.g., cluster-based [2, 17, 31, 33] and peer-to-peer [1, 13, 20] storage systems) attempt to offer cost-effective, reliable data stores on top of unreliable, commodity or even donated storage components. To tolerate failures of individual nodes, these systems use data redundancy through replication or erasure coding. This approach faces two problems. First, regardless of the redundancy level used, there is always a non-zero probability of a burst of correlated permanent failures up to the redundancy level used; hence the possibility of permanently losing data always exists. Second, data loss probability increases with the data volume stored when all other characteristics of the system are kept constant.

This paper explores the feasibility of a novel storage architecture that offers the access throughput and the reliability characteristics of an enterprise-class system at a much lower price point. This appealing cost/performance point can be obtained by exploiting two opportunities:

- First, in today's organizations, the storage capabilities of LAN-connected workstations are often underused: disks have vast amounts of idle space, and the corresponding I/O channels rarely serve data at full capacity [1]. This creates an opportunity to 'scavenge' storage; that is, to aggregate the storage space and I/O bandwidth of these network-connected machines to build a low-cost, high-performance, data-store.
- Second, data reliability can be split into two interrelated components: durability (i.e., ability to preserve data over time) and availability (i.e., ability to instantly serve the data). For many applications, durability is the critical property: they may tolerate short-term service interruptions (that is, lower availability) as long as data is not permanently lost (e.g., due to disk failure). Decoupling durability and availability offers the opportunity to engineer systems that provide strong durability guarantees (e.g., ensuring ten 9's durability) while relaxing availability guarantees to reduce the overall cost.

To obtain high I/O throughput and strong durability while keeping costs under control, we propose an architecture that integrates two types of low-cost storage components. First, *durable nodes*: a set of dedicated, yet with low I/O bandwidth (thus cheap) nodes that provide data durability. For example, this component can be an Automated Tape Library (ATL) or a storage utility such as Amazon S3 [5]. Second, *volatile nodes*: a large number of volatile nodes that, on aggregate, provide a low-cost, high throughput storage space. These nodes, for example, can be a subset of the desktops available in a company or a research institution.

One usage scenario for this hybrid architecture is a GridFTP server. To enable high I/O access rates required by scientific applications, GridFTP [3] deployments are often supported by massive hardware resources (e.g., clusters and parallel file systems). Integrating

HPDC'09, June 11–13, 2009, Munich, Germany.

GridFTP with a combination of dedicated and scavenged storage resources offers the opportunity to lower GridFTP deployment cost while maintaining high performance. Another usage scenario is a high-performance data-store geared towards a read-mostly workload that uses a storage utility (e.g., Amazon's S3) for data durability, and local workstations as the front-end for high-performance data access.

Integrating these components brings two challenges: First, *correctly provision the controlled resources to provide user-defined guarantees*. In this architecture, the scarce resource is the bandwidth of the durable component(s), which may limit data availability in the entire system. Other factors that affect availability include the number of objects maintained by the system, the replication level, and the characteristics of the volatile nodes (e.g., failure and repair times, and bandwidth). The goal of this paper is to understand the relationship between these factors and availability in order to offer tools to correctly provision system's resources to provide user-defined availability guarantees.

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 system-wide availability as well as to minimize the amount of data that needs to be restored using the durable component. This last point is essential when outsourcing the durable components to an external storage utility that may charge according to the volume of data transferred (like Amazon S3 currently does).

To tackle these challenges, this paper addresses the following questions:

- Consider a system built atop of volatile, unreliable components only, that 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, replication level)?*
- In the system considered above, if we take one of the replicas and place it on a durable component with low access bandwidth (e.g., a tape), *what is 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, mean time to repair obtained by characterizing the environment), *what is the impact of resource constraints (e.g., bandwidth at the durable and volatile nodes) on the resulting availability level and, consequently, the volume of traffic generated at the durable and volatile nodes?*
- Once the system and workload characteristics are fixed (e.g., bandwidth constraints, failure rates), *what replica placement scheme enables maximum availability?*

To answer these questions, we present an analytical model based on Markov chains. This model captures the key relationships between availability and the main system characteristics: the repair and failure rates of the volatile nodes and the repair rate of the durable components. Further, to study the impact of other factors that can not be included in the analytical model as it would make it intractable (like, for example, transient failures, the details of replica placement scheme, and the detailed characteristics of the workload and the deployment environment), we develop a low-level simulator for the proposed architecture. The simulator uses as inputs machine availability traces, implements the details of the replication scheme, and provides an accurate prediction of the system state. Finally, to demonstrate feasibility and evaluate performance, we integrate the Globus' project GridFTP server [4] and MosaStore [2], a data store based on scavenged storage.

The contributions of this work are as follows:

- First, we propose a low-cost, reliable, high-performance data-store architecture. We explore the tradeoffs compared to a pure replication-based solution using analytical modeling and simulations, and we demonstrate its feasibility through a prototype implementation that confirms the high-throughput properties targeted.
- Second, we provide tools that allow for dimensional analysis and system provisioning of the proposed architecture: an analytical model that captures the main system characteristics, and a simulator that allows for detailed performance predictions.
- Third, we evaluate the impact on availability for three replica placement schemes. We determine that creating a new replica at the node that offers 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).

We note that 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 on the durable component. Regardless of the approach used to handle writes, our study enables a reduced read load on the durable component which, in turn, helps improve the write throughput by leaving a higher share of the durable component throughput available for the write load.

The rest of this paper is organized as follows. Section 2 presents background and related work. Section 3 presents the system model. Section 4 analyses the relationship between the system's characteristics and data availability through an analytical model (Section 4.1) and simulations (Section 4.2). Section 5 evaluates a use case application (a GridFTP server). Section 6 concludes.

2. BACKGROUND AND RELATED WORK

A large number of storage systems use replication or erasure codes to improve reliability without differentiating between data durability and availability. *Peer-to-peer (p2p) storage systems* (e.g., OceanStore [20], TotalRecall [6], Ivy [24], Fariste [1], PAST [14], and CFS [13]), for example, harness idle disk space from thousands of wide-area, distributed workstations and use replication to both reduce the probability of losing data and increase availability. Similarly, *cluster-based storage systems* (e.g., GFS [17], Ceph [33], PVFS [10]) hosted on dedicated, well-connected components use the same techniques, yet operate in a different deployment environment.

Closer to our system are *scavenged storage systems* (FreeLoader [31], dCache [15], MosaStore [2]) which aggregate idle storage space from LAN-connected workstations. For example dCache [15] combines heterogeneous disk-based storage systems (including donated storage) to offer a unified, high-performance data store.

dCache enables the inclusion of a tertiary storage system as back-end data store. dCache designers, however, do not provide guidance on provisioning the system to provide user-defined guarantees for data availability and durability.

The differentiating factors that drive the design of these systems are the characteristics of the deployment environment: the maintenance bandwidth available at the participating nodes as well as their failure and repair rates. Blake et al. [7] argue that bandwidth constraints ultimately determine the achievable storage capacity for given data reliability targets regardless of the replication level used. In these conditions two techniques have been widely used to reduce the volume of generated traffic:

- *Separating durability and availability:* Chun et al. [11] decouple durability and availability to reduce maintenance costs in the presence of failures. Lefebvre et al. [21] provide a formula for determining the replica repair rate when durability is the only concern, and show that targeting availability requires much more frequent repairs.

  Our approach is different in that we completely separate durability and availability management. Durability is related to the characteristics of the durable component, while the desired level of availability is controlled by correctly provisioning the system (e.g., the bandwidth of the durable and volatile nodes). This enables a higher degree of flexibility to decide on the availability level without affecting durability.

- *Differentiating between transient and permanent failures* enables generating lower maintenance traffic. Carbonite [11] 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 maintenance cost on the volatile nodes.

  Another approach to mask transient failures, suggested by Blake et al. [7], is to use long timeouts when detecting node failures. Although this technique reduces the number of unnecessary repairs, it increases the time to repair a misclassified permanent failure, therefore threatening durability.

Finally, compared to Hierarchical Storage Management (HSM) systems [9], we make two contributions: first we integrate a set of volatile resources as the frontend, and second, we provide the tools to adequately provision these systems based on the characteristics of their components.

Analytical Models of Replicated Systems. Markov chains have been used in the past to model replication-based systems. Chun et al. [11] use Markov chains to derive an expression for the number of expected replicas of an object for a given creation and failure rates. Ramabhardan et al. [28] evaluate the expected object lifetimes in a distributed storage system and study the impact of resource constrains. A similar model is also considered in [32], [12].

Our analytical model is based on these efforts. We expand Chun et al. [11] model to add the durable component and derive an expression for data object availability.

3. SYSTEM MODEL

This section presents the high-level design of the system and the precise definition of availability used throughout the paper.

We model the system as having one durable node and a large number of volatile nodes. In reality, the durable node could be an external archival service or a complex set of devices (e.g., a tape archive); however, for the purpose of our analysis, we model it as a single durable component.

The system operates under the following assumptions:

- *The durable component maintains a copy of all objects in the system*. This forms a backend data store that recreates the data that the volatile nodes may lose, hence offering strong durability guarantees for the whole system.
- *The clients access the system only through the volatile nodes*. If the data is not available on the volatile nodes (e.g., in case of a crash that hits all replicas of a data object), then 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. The rationale not to use the durable node to serve application's read requests is to minimize the cost of the durable node by minimizing the generated I/O read load.
- *Logically centralized metadata service*. Clients locate objects by contacting a metadata service. This service maintains all metadata related to objects and makes replica placement decisions. Note that clients never read or write data through the metadata service, hence keeping it outside the critical I/O path.
- *Failures are detected using timeouts*. Volatile nodes declare their availability to the system using heart-beat messages sent to the metadata service.
- *The system handles two types of failures: transient and permanent failures*. Permanent failures cause data loss (e.g., a disk failure). 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 dealing with transient failures, replicas are reintegrated once they re-join the system.
- *Replica repair model*. The system seeks to maintain a minimum replication level $\boldsymbol{n}$ for all objects at all times. Once a failed volatile node is detected, the system reacts by creating additional copies to increase the replication level back to $\boldsymbol{n}$. Each volatile node is allowed a limited amount of repair bandwidth $\boldsymbol{b}$. If the replication level on the volatile nodes goes down to $\mathbf{0}$ for a specific object, the durable component creates, with a repair bandwidth limited to $\boldsymbol{B}$, 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 $\boldsymbol{n}$.
- *Security Model*. We assume a trusted deployment environment in which all system components and the communication channels among them are trusted. This assumption, however, does not limit the generality of our work: security in distributed storage have been addressed widely in the literature [16, 22].

Defining availability. Under these assumptions, although objects are never lost due to the contribution of the durable component, an object may not be instantly accessible if it is not available on at least one volatile node. Therefore, in this context, *we define availability as the ratio of the total time an object replica exists on at least one volatile node*.

4. AVAILABILITY STUDY

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

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

4.1 The Analytical Model

This section describes our analytical model (§ 4.1.1), derives an expression for object availability (§ 4.1.2), and discusses the accuracy of the model (§ 4.1.3).

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.

Figure 1 illustrates the model. An object can have n maximum 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 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 with rate λ. This repair rate λ depends on the volatile nodes' repair bandwidth and the size of the data object.

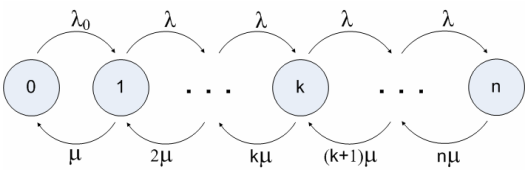

Figure 1: Markov chain model. Each state represents the number of currently available replicas on the volatile nodes.

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 that the failure rate μ and the repair rates λ_k are exponentially distributed. 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).

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 [19]:

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

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

$$\mu_k = k\mu \qquad k \ge 0$$

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. Additionally we will refer to the term λ_0/μ as γ, which intuitively 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). Finally, when $n=1$, availability depends only on the durable node repair ratio γ, which is intuitive.

4.1.3 Model Accuracy

This section summarizes the limitations of the analytical model presented above. Table 1 presents the main factors that affect the system state, and the way the analytical model and the simulations (discussed in the next section) capture them.

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, since the system is built atop of donated storage, nodes may frequently depart and rejoin the system without actually losing data.

Second, *the model assumes exponentially distributed replica repair and life times*. Empirical evidence supports the assumption that permanent failures (e.g., due to disk failure) are well modeled by an exponential distribution [12, 28, 29]; this assumption, however, is not well supported for replica repair times.

Third, *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 a fixed replica repair rate irrespective of the number of concurrent

Table 1: A summary of factors that affect system state and the way they are captured by the analytical and simulation models

Factor	*Analytical model*	*Simulation model*
Transient failures.	Not modeled	Trace-driven.
Permanent failures.	Exponentially distributed failure time.	Trace-driven.
Number of objects and their placement on the volatile nodes.	The model is agnostic to the number of objects maintained by the system and the placement policy as it studies the state of a single object only.	Replicas are distributed according to a specific placement policy which is accurately simulated.
Replica repair rate (in case more than one replica is needed to obtain the target replication level ***n*** for a specific object).	Replicas are repaired sequentially one at a time. For example, if the target replication level is four and an object has two available replicas, the missing replicas are created sequentially; however, in reality, the system could create both replicas in parallel without reducing the creation time.	Replicas of the same object are repaired in parallel according to the number of existing replicas. Further, replication requests are queued at each node, and served sequentially according to the node's available repair bandwidth.
Bandwidth limits for replica repair at the durable and volatile nodes.	Modeled indirectly as a part of the exponentially distributed repair time. In reality, the repair time is associated with bandwidth and data size, and is not likely to vary greatly.	Fixed maximum rate**.** The simulator also models contention on a node's access link by queuing replica creation operations.
Replica management overhead (e.g., time to detect a failure, decide where to get a replica from and where to place it).	Not part of the model.	Constant time. This is acceptable as long as in the real world management overhead is at least one order of magnitude lower than replica repair time.
Correlated replica failures.	Replicas are assumed to fail independently; however, in reality, multiple replicas exist on the same node, hence failures are correlated.	Exact placement is modeled with multiple replicas of different objects on the same node, hence replicas may fail simultaneously.

repairs. Another implication of this limitation is that the model does not capture the effect of replica placement decisions.

Fourth, *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 remaining $n-k$ 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 λ*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). Therefore, the model can not predict data availability after a specific elapsed time.

In spite of these limitations, the analytical model offers a compact closed-form analytical expression that:

- unveils the key relationships between system characteristics, hence it is still useful for a coarse-grain, qualitative characterization of the system and,
- 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.

4.2 The Simulation Model

We use SimPy [30], a process-based discrete-event simulation language, to build the simulator. This section describes the simulation model (§ 4.2.1), discusses the accuracy of the model (§ 4.2.2), and how we validate the simulator (§ 4.2.3).

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 disk space.

The simulator is driven by: (i) a trace of failure events (transient and permanent) for the volatile nodes (the traces are discussed below), and (ii) 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 polices. The simulator monitors the state of the objects (i.e., the current replication level), and the amount of traffic generated by all nodes throughout the simulation.

When the metadata service detects a volatile node failure (note that multiple volatile nodes may fail at the same time step), it attempts to increase the replication level of all objects maintained by the failed node back to the minimum replication level n. For each lost object copy, 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). A repair request is processed by a source storage node 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 employed replica placement scheme. 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 acknowledged in FIFO order; as a result, a node's download channel is not time-shared as it is dedicated for one transfer at a time.

4.2.2 Model Accuracy

Table 1 summarizes the improvements of the simulation model over the analytical model. While the simulation model is more realistic than the analytical model presented, there are two limitations left.

First, the model assumes no shared bottlenecks in the network core and that each node can directly contact all other nodes. Given that we target a LAN environment with good connectivity and high-speed core switches, this assumption is acceptable.

Second, the simulator is driven by traces of node failure and repair events, thus its accuracy depends on the accuracy with which these traces reflect real-world events. On the one side, we use Weibull distributions to generate transient failures inter-arrival times. Heath et al. [18] demonstrated that Weibull distributions model well desktop availability in an enterprise environment, Nurmi et al. [25] also reached to the same conclusion. On the other side, exponential distributions are widely accepted to model well permanent failure inter-arrival times [11, 12, 28, 32].

We use synthetic traces instead of real-life traces for two reasons. First, to the best of our knowledge, there are no publically available long enough traces (in the order of years) for machine availability for the environment we target (i.e., LAN-connected machines in an enterprise network). For example, the traces analyzed by most machine availability studies, such as [8, 18, 25], are at most in the order of few months. Second, synthetic traces enable us to increase failure density, hence allowing for better investigation of the key system trends.

4.2.3 Validating the Simulator

To increase our confidence in the correctness of the simulator we:

- *Compared simulation results with those predicted by the analytical model.* As we demonstrate in the next section, the simulator and the analytical results are close for cases where the limitations of the analytical model do not preclude a tight estimation of availability.
- *Manually verified the state of the simulator.* We ran small-scale configurations (i.e., few nodes and objects) and recorded the state of all elements (both nodes and objects) across time. We then verified these records manually against the expected behavior of the simulator for a specific placement scheme.
- *Followed good programming practices.* The simulator code includes asserts for state correctness whenever a new event occurs.

4.3 Simulation-based Evaluation

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

Unless otherwise noted, simulations are configured as follows: the system maintains 32 TB of data divided into 2GB objects. The replication level n is set to 3, a common value used in distributed storage systems [17]. Since the impact of the object size is that small objects increase the opportunity for parallel repair thus increase availability; our assumption to use large objects is a conservative one. A more detailed discussion on the impact of object size appears in [27].

The system contains 500 volatile nodes, each allocating up to b = 2Mbps for replica maintenance. The durable node offers up to B = 1Mbps read throughput. This value is within the range of the wide-area download throughput offered by Amazon S3 [26].

Further, we use four year-long synthetic traces with the following parameters. For transient failures, the Weibull distribution parameters are: 0.49 for the shape factor, and 10 days for the scale factor (similar to [18], and the corresponding node MTTF is 20 days). For permanent failures, the exponential distribution has a node MTTF of one year (similar to [11]).

Finally, we report unavailability rather than availability as it is easier to plot on logarithmic scale. For all experiments, we report averages over 30 runs (availability in each run is the average availability of all objects in the system) and 95% confidence interval.

4.3.1. Comparing the analytical model with simulation.

The first question addressed by our experiments is: *how close the analytical model is to simulations?* To answer this question, we estimate unavailability for various values of volatile nodes' bandwidth. Figure 2 shows that the analytical model and the simulation results are close.

Two details of this experiment are worth discussing: First, as mentioned in section 4.1.3, the analytical model does not model transient failures; therefore, in this simulation, we only use the permanent failures trace.

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 failure rate μ, and the creation rates λ and λ_0 implied in the traces used.

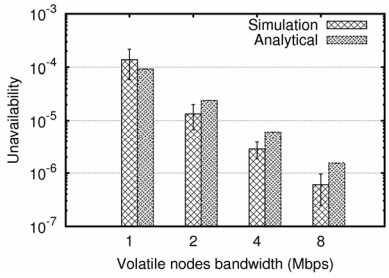

Figure 2: Comparing unavailability predicted by the analytical model and simulations.

On the one hand, the replica failure rate μ depends on the volatile nodes' failure rate. As 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. On the other hand, the replica creation rate λ depends on the amount of data that needs 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. This means that 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 $2*b/d$. In this experiment, there are 500 volatile nodes, and the average amount of data maintained by each node is d = 32*3/500TB ≈ 196GB; Thus, for example, for the first data point where b=1Mbps, $\lambda \approx 39$ replica copies per year. The same estimation also applies to λ_0 by replacing b with the durable node's repair bandwidth.

4.3.2. Storage load vs. durability.

Consider a system built atop of unreliable, volatile components only. This system employs replication to provide both durability and availability. To compare with our system that uses a durable component, we answer the following question: *What is the maximum amount of data that can be preserved durably during a specific time interval, and 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. Figure 3 shows the fraction of objects lost at the end of the experiment.

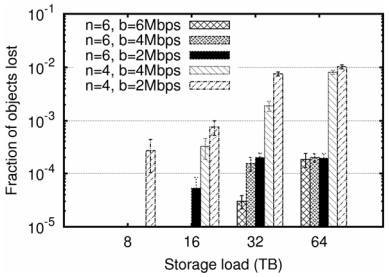

Figure 3: The fraction of objects lost after four years. (Missing bars correspond to zero loss rates). When aggressively increasing the replication level and the bandwidth limits (n = 8 and b = 8Mbps), the system does not lose any objects for storage load up to 64TB.

As the storage load grows, each node maintains a higher data volume. Consequently, repairing a failed node data 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 example, when using $\boldsymbol{n}$ = 8 and $\boldsymbol{b}$ = 8Mbps, the system does not lose any objects while maintaining up to 64TB storage load. However, increased durability comes at the cost of increased maintenance traffic and disk space. Further, when the storage load increases beyond 64TB, 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 replication 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 this end, we run an experiment comparing the two systems. For the system with durable component, we use our default configurations: $\boldsymbol{B}$ = 1Mbps, $\boldsymbol{b}$ = 2Mbps and $\boldsymbol{n}$ = 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 64TB: $\boldsymbol{b}$ = 8Mbps and $\boldsymbol{n}$ = 8 replicas.

Figure 4 demonstrates that the system with the durable component generates 50% less maintenance traffic than the system without durable component. This also corresponds to 50% less average amount of bandwidth consumed over the 4 year simulation period. For example, for storage load of 64TB, the system without durable component generates on average an aggregate traffic of about 140Mbps, a significant burden even for a gigabit LAN compared to only 65Mbps for the system with durable component. Finally, the system with a durable component requires significantly less disk space as it uses 4 replicas (1 on the durable node and 3 on the volatile nodes) compared to 8 replicas used by the system without a durable component.

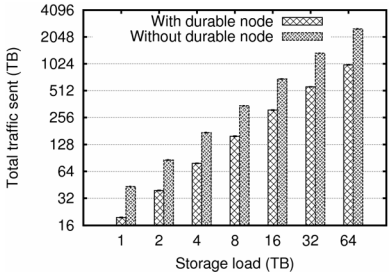

Figure 4: Total traffic sent at end of experiment (log scale).

The price paid for these savings is twofold: a slightly more complex, asymmetric system that includes the durable component, and slightly lower availability. Figure 5 plots data unavailability for the system with durable component. Unavailability increases as storage load increases; still, the system is able to provide acceptable availability levels even for large storage volumes.

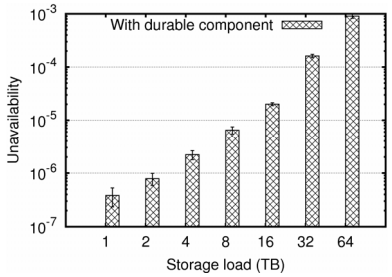

Figure 5: Unavailability (the lower the better) while varying the volume of data stored.

Note that for the system without a durable component, durability and availability are in effect similar and, for the parameters we consider, the system offers full availability. However, this is at the expense of extra storage and maintenance traffic.

4.3.4. Provisioning system resources

The next question we consider is: *What is the effect of bandwidth limits at the durable or the volatile nodes on the achieved availability and on the generated network traffic?*

To answer this question we run a simulation while varying the bandwidth of the durable component. Figure 6 shows that increasing the bandwidth of the durable component does not dramatically enhance availability. For example, in this experiment, reducing unavailability by one order of magnitude requires roughly increasing the durable component's bandwidth 16 fold while keeping the volatile node's bandwidth constant.

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. As the figure shows, unavailability decreases more noticeably as the volatile node's bandwidth increases. For example, reducing unavailability by one order of magnitude requires increasing the volatile nodes'

bandwidth only 4 fold while keeping the durable component's bandwidth constant.

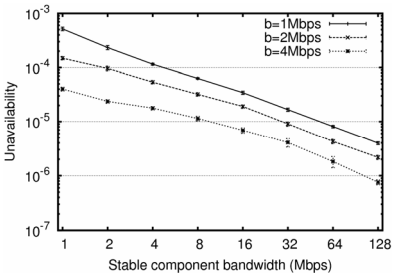

Figure 6: Unavailability (the lower the better) while varying the bandwidth of the durable component *B*.

Figure 7 illustrates the total amount of traffic generated at the durable component and at the volatile nodes for the same experiment. We note that, although it results in better availability, increasing the stable component bandwidth does not result in proportionally higher generated maintenance traffic. This is an important result if we consider that the durable component may be an outsourced storage utility which may charge based on the volume of generated traffic like Amazon S3 does.

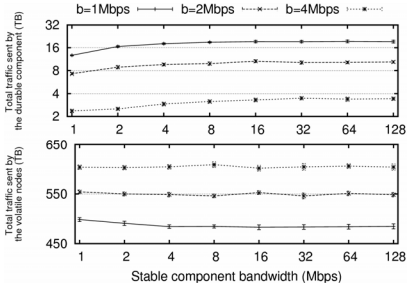

Figure 7: Cumulative traffic generated at the end of the trace. Top plot (logarithmic Y axis): traffic sent by the durable component. Bottom (linear Y axis): traffic at the volatile nodes.

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.

Figure 8 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 maintenance 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.

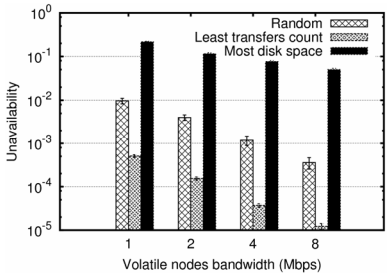

Figure 8: Comparing different replica placement schemes.

5. USE CASE: A GRIDFTP SERVER

While the previous section focused on exploring the relationships between data availability and various system characteristics in the context of the architecture we propose, this section aims to explore the practical feasibility of this architecture and demonstrates its high I/O throughput characteristics.

GridFTP [3] has become the data access protocol of choice for data-intensive scientific communities. As a result, significant efforts have been made to optimize the protocol itself and the software stack implementing it – for example, the protocol incorporates new features (e.g., striping and parallel transfers) to take advantage of 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 enable high I/O access rates (e.g., clusters and parallel file systems). We aim to demonstrate that our approach can reduce the cost of GridFTP deployments while maintaining their high-performance characteristics.

5.1 Solution and Components

Space constraints allow us to only briefly describe our prototype that integrates two systems (Figure 9):

- *Globus' project GridFTP framework* [4]: 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 the underlying storage system (Figure 9). First, the *control interface* integrated with the GridFTP protocol interpreter (PI) module to form the *server PI*. This module parses the FTP commands received on the control channel, and calls the appropriate handlers from the control interface. Second, the *data interfac*e controls the Data Transfer Process (DTP) which handles access to the stored data and its transfer through the data channel.

  To accelerate data transfers, the GridFTP framework implements *striped* data transfers. This feature enables the establishment of parallel data channels between pairs of storage nodes. A striped GridFTP server uses at least one *Server PI* that is running on a designated node known by the clients, and one DTP process on each storage node.

- *MosaStore* [2]: a highly configurable scavenged storage system designed to harness unused storage space from LAN-connected machines. MosaStore consists of two main components: a

metadata service and the benefactor nodes that donate storage space. To increase performance, MosaStore employs striping: files are divided into smaller chunks distributed among the volatile benefactors. The metadata service maintains information related to the available space, 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-/off-line, and available disk space) to the management service, and serve client requests to store/retrieve data chunks. Finally, a client performs a read operation by asking the metadata service for a file's chunks-benefactors mapping. Once obtained, the client pulls the data directly from the benefactors.

The main challenge relates to the transparent integration of these two systems. Figure 9 presents the integrated architecture and outlines the new components we added. A GridFTP/MosaStore integrated server consists of three main components:

- *The server PI* parses GridFTP commands sent by the client, and invokes the GridFTP/MosaStore control interface to handle the requests. Depending on the command, the control interface may query MosaStore's metadata service, or delegate the handling to the DTPs running on the benefactors.
- *MosaStore's metadata service* resembles the centralized metadata service in our proposed architecture. It maintains the system's metadata, detects failed nodes, and makes replica placement decisions.
- *The benefactor nodes* represent the volatile nodes in the system. The benefactors handle replication requests sent by the metadata service, and run the GridFTP DTP component which handles data access requests from the server PI.

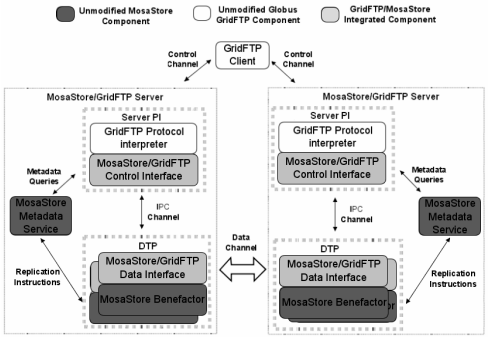

Figure 9: GridFTP/MosaStore Integration in the context of third-party transfer.

In the following we use a third-party transfer scenario to present the interaction of these components. The 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 contacts the metadata service asking for the chunk-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. Next, 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 addresses obtained previously from the source server. The PI at the destination server invokes the control interface passing it the data source addresses. The control interface at the destination server contacts the metadata service asking for a set of benefactors that may provide the required storage space. Once received, 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 benefactors of the source server. Once the data is written to the destination benefactors, the background replication daemon, invoked by the metadata service, creates the necessary number of replicas on other benefactors as well as on the durable component.

Finally, we note that the current status of our prototype does not take into account the durable component, which can be easily included by modifying the replica placement decision mechanisms at the metadata service. Not including the durable component, however, has no impact on the transfer throughput evaluation presented in this section as the durable component has no role in serving data to the clients (recall that data is served only by the front-end volatile nodes).

5.2 Evaluation

We evaluate our prototype on a testbed of 22 nodes each with a 2.33GHz Quad-core CPU, 4GB memory and connected at 1Gbps. We compare our server with unmodified GridFTP servers running over NFS [23] and PVFS [10].

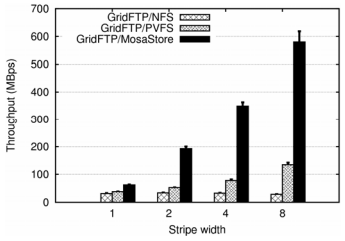

Figure 10: Average achieved throughput while transferring 1GB file, and varying the stripe width.

The first experiment measures the achieved throughput for a third-party transfer between two identical servers (e.g., both GridFTP/MosaStore). The left bars in Figure 10 show that the GridFTP/NFS throughput does not vary with the stripe width. The reason is the NFS share which forms a bottleneck at both ends. The middle bars present the performance of the GridFTP/PVFS server. In this setup, each DTP process accesses PVFS through a client module that offers a traditional file system interface; hence, introducing an extra level of indirection at both the source and destination servers, the main reason for still limited throughput.

The GridFTP/MosaStore server, however, directly integrates the DTP processes with MosaStore's benefactor nodes. This offers full parallelism since the storage nodes at both ends are connected directly to each other without shared bottlenecks or an extra level of indirection. As a result, our integrated server achieves up to 300% increase in throughput compared to GridFTP/PVFS server.

Finally, we evaluate the performance of our solution under high load: we setup a GridFTP server supported by 10 storage nodes, and accessed by 40 clients running on 10 machines. Each client reads 100 files of 100MB each.

In this experiment, GridFTP/Mosastore server was able to support 800MBps aggregate sustained throughput compared to only 500MBps achieved by GridFTP/PVFS setup. This illustrates the ability of our system to efficiently harness available bandwidth and to scale to support an intense workload. Further, it demonstrates that the effort to integrate GridFTP and MosaStore to streamline the data-path at low level pays off: 60% increase in aggregate

throughput compared to out-of-the-box (still optimized in terms of configuration parameters) GridFTP/PVFS setup.

6. SUMMARY

This study demonstrates the feasibility of a low-cost storage architecture that offers the durability and access performance characteristics of a well-endowed system. The proposed architecture is based on (*i*) integrating large number of volatile components and low-bandwidth durable components, and (*ii*) decoupling data durability and availability, which enables relaxing availability guarantees to a level acceptable by applications, yet providing strong durability to reduce the cost of data maintenance.

We presented analytical and simulation-based tools that can be used to provision systems based on the architecture we propose, and to evaluate the effect of deployment environment characteristics, workload characteristics, and configuration choices (e.g., replication level) on the resulting system properties (e.g., availability and generated maintenance traffic). Further, we compared the performance of the architecture we propose with that of systems based purely on replication: we show that adding the back-end durable component halves the generated maintenance traffic as well as the disk space used, while offering the same data durability. These savings come at the cost of slightly lower availability and the additional complexity of an asymmetric data replication scheme. Further, we evaluate the impact of various replica placement schemes on data availability. We determine 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 replica placement.

Finally, through a prototype use-case application, we demonstrate that our approach can reduce the cost of data-access infrastructures while maintaining their high-performance characteristics.

Acknowledgements: We thank Samer Al-Kiswany, Raj Kettimuthu and Michael Link for their insightful comments on earlier versions of this paper and/or help with GridFTP code.

7. REFERENCES

[1] Adya, A., et al. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. OPERATING SYSTEMS REVIEW, 36(2002), 1-14.

[2] Al-Kiswany, S., et al. stdchk: A Checkpoint Storage System for Desktop Grid Computing. In ICDCS, 2008.

[3] Allcock, W., et al. GridFTP: Protocol Extensions to FTP for the Grid. Global Grid ForumGFD-RP, 20(2003).

[4] Allcock, W., et al. The Globus Striped GridFTP Framework and Server. In Supercomputing, 2005.

[5] Amazon Web Services. http:/s3.amazonaws.com.

[6] Bhagwan, R., et al. Total recall: system support for automated availability management. In NSDI, 2004.

[7] Blake, C. and Rodrigues, R. High availability, scalable storage, dynamic peer networks: pick two. In HOTOS, 2003.

[8] Bolosky, W. J., et al. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In SIGMETRICS, 2000.

[9] Brubeck, D. and Rowe, L. Hierarchical storage management in a distributed VOD system. IEEE Multimedia, 3, 3 (1996), 37-47.

[10] Carns, P. H., et al. PVFS: a parallel file system for linux clusters. In ALS, 2000.

[11] Chun, B., et al. Efficient replica maintenance for distributed storage systems. In NSDI, 2006.

[12] Dabek, F. A Distributed Hash Table. Massachusetts Institute of Technology, 2005.

[13] Dabek, F., et al. Wide-area cooperative storage with CFS. ACM SIGOPS Operating Systems Review, 35, 5 (2001), 202-215.

[14] Druschel, P. and Rowstron, A. PAST: A large-scale, persistent peer-to-peer storage utility. In HotOS, 2001.

[15] Fuhrmann, P. dCache, the commodity cache. In proceedings of the Twelfth NASA Goddard and Twenty First IEEE Conference on Mass Storage Systems and Technologies, Washington DC., 2004.

[16] Gharaibeh, A., et al. Configurable security for scavenged storage systems. In StorageSS, 2008.

[17] Ghemawat, S., et al. The Google file system. In SOSP, 2003.

[18] Heath, T., et al. The Shape of Failure. In Proceedings of the First Workshop on Evaluating and Architecting System dependabilitY (EASY). 2001.

[19] Kleinrock, L. Theory, Volume 1, Queueing Systems. Wiley-Interscience, 1975.

[20] Kubiatowicz, J., et al. OceanStore: an architecture for global-scale persistent storage. ACM SIGARCH Computer Architecture News, 28, 5 (2000), 190-201.

[21] Lefebvre, G. and Feeley, M. J. Separating durability and availability in self-managed storage. In EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop.

[22] Leung, A. W., et al. Scalable security for petascale parallel file systems. In Supercomputing, 2007.

[23] Microsystems, S. and others. NFS: Network file system protocol specification. Request for Comments, 1094(1988).

[24] Muthitacharoen, A., et al. Ivy: a read/write peer-to-peer file system. In OSDI, 2002.

[25] Nurmi, D., et al. Modeling Machine Availability in Enterprise and Wide-Area Distributed Computing Environments. LECTURE NOTES IN COMPUTER SCIENCE, 3648(2005), 432.

[26] Palankar, M. R., et al. Amazon S3 for science grids: a viable solution? In DADC '08: Proceedings of the international workshop on Data-aware distributed computing.

[27] Qiao Lian, et al. On the Impact of Replica Placement to the Reliability of Distributed Brick Storage Systems. In ICDCS, 2005.

[28] Ramabhadran, S. and Pasquale, J. Analysis of Long-Running Replicated Systems. In INFOCOM, 2006.

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

[30] Simpy homepage. http://simpy.sourceforge.net/ (2009).

[31] Vazhkudai, S. S., et al. FreeLoader: Scavenging Desktop Storage Resources for Scientific Data. In Supercomputing, 2005.

[32] Weatherspoon, H. Design and Evaluation of Distributed Wide-Area On-line Archival Storage Systems. University of California Berkeley Ph.D.Dissertation, (2006).

[33] Weil, S. A., et al. Ceph: a scalable, high-performance distributed file system. In OSDI, 2006.