EXPLORING DATA RELIABILITY TRADEOFFS IN REPLICATED STORAGE SYSTEMS

by

Abdullah Hassan Gharaibeh

B.Sc., Jordan University of Science and Technology, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate Studies

(Electrical and Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

June 2009

© Abdullah Hassan Gharaibeh, 2009
Abstract

This thesis 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: a GridFTP server that aggregates volatile resources. Our evaluation demonstrates an impressive, up to 800MBps transfer throughput for the new GridFTP service.
# Table of Contents

Abstract ........................................................................................................................... ii

Table of Contents ........................................................................................................... iii

List of Tables ................................................................................................................... v

List of Figures ................................................................................................................ vi

Acknowledgements ....................................................................................................... vii

Dedication ..................................................................................................................... viii

1. **Introduction** ........................................................................................................ 1
   1.1. Motivation ........................................................................................................ 1
   1.2. Opportunities and Solution .............................................................................. 2
   1.3. Challenges ........................................................................................................ 4
   1.4. Methodology .................................................................................................... 5
   1.5. Contributions ................................................................................................... 6
   1.6. Thesis Structure ............................................................................................... 7

2. **Background and Related Work** ........................................................................ 8
   2.1. Redundant Storage Systems ............................................................................ 8
       2.1.1. Peer-to-peer (p2p) Storage Systems ...................................................... 10
       2.1.2. Cluster-based Storage Systems .............................................................. 12
       2.1.3. Scavenged Storage Systems .................................................................. 13
   2.2. Analytical Models of Replicated Storage Systems ........................................ 14

3. **System Model** .................................................................................................... 16
   3.1. Components ................................................................................................... 16
   3.2. Assumptions .................................................................................................. 17
   3.3. Discussion ...................................................................................................... 18

4. **Availability Study** ............................................................................................ 21
   4.1. The Analytical Model ..................................................................................... 21
       4.1.1. The Model .............................................................................................. 21
       4.1.2. An Expression for Object (Un)availability ........................................... 22
       4.1.3. Model Accuracy ...................................................................................... 24
   4.2. The Simulation Model .................................................................................... 26
       4.2.1. The Model .............................................................................................. 26
       4.2.2. Model Accuracy ...................................................................................... 27
       4.2.3. Validating the Simulator ........................................................................ 29
   4.3. Simulation-based Evaluation .......................................................................... 30
       4.3.1. Comparing the Analytical Model with Simulation ............................... 31
       4.3.2. Storage Load vs. Durability ................................................................. 33
List of Tables

Table 1: $p_0$ for a few small values of replication level (n)................................. 24
Table 2: A summary of factors that affect system state and the way they are captured by
the analytical and simulation models. ................................................................. 28
Table 3: Statistics of aggregate bandwidth consumed for systems with and without
durable component ............................................................................................ 36
Table 4: Statistics of unavailability for the system with durable component while
varying the volume of data stored..................................................................... 37
Table 5: Statistics of unavailability while varying the durable node’s bandwidth (B). 38
Table 6: Statistics of total aggregate replication bandwidth consumed at the durable
node (Mbps) while varying the durable node’s bandwidth (B)......................... 39
Table 7: Statistics of unavailability while varying the volatile nodes’ bandwidth (b)... 41
Table 8: Statistics of aggregate replication bandwidth consumed (Mbps) while varying
the volatile nodes’ bandwidth (b)..................................................................... 41
Table 9: Statistics of unavailability while varying the replication level.................... 42
Table 10: Statistics of aggregate replication bandwidth consumed (Mbps) while varying
the replication level .......................................................................................... 43
List of Figures

Figure 1: System high-level architecture................................................................. 16
Figure 2: Markov chain model................................................................................ 22
Figure 3: Unavailability predicted by the analytical model and simulations............. 32
Figure 4: Fraction of objects lost at end of simulation............................................ 33
Figure 5: Total traffic sent for systems with and without durable component............ 35
Figure 6: CDF of total replication bandwidth (with and without durable component)... 36
Figure 7: CDF of unavailability for the system with durable component while varying the volume of data stored.......................................................... 37
Figure 8: CDF of unavailability while varying the durable node’s bandwidth (B).... 39
Figure 9: CDF of aggregate replication bandwidth while varying the durable node’s bandwidth (B). .................................................................................. 40
Figure 10: CDF of unavailability while varying the volatile nodes’ bandwidth (b). .... 41
Figure 11: CDF of aggregate replication bandwidth consumed while varying the volatile nodes’ bandwidth (b). ................................................................. 42
Figure 12: CDF of object unavailability while varying the replication level n......... 43
Figure 13: CDF of total aggregate replication bandwidth consumed while varying the replication level (n). ............................................................................... 44
Figure 14: Average unavailability for three different replica placement schemes...... 45
Figure 15: High-level architecture of the GridFTP use case.................................... 47
Figure 16: A typical FTP server design (left), Globus’s GridFTP server design in a typical deployment (right) in the context of a third-party transfer............... 50
Figure 17: GridFTP/MosaStore integrated architecture........................................... 55
Figure 18: GridFTP server average achieved throughput ......................................... 59
Figure 19: GridFTP server scalability ..................................................................... 60
Acknowledgements

I would like to sincerely thank my supervisor, Professor Matei Ripeanu, for his consistent, honest and insightful feedback in the last two years. This work would not have been possible without his guidance and support.

I am also thankful to a number of special friends who made my graduate school journey a rich and exciting experience: Samer Al-Kiswany and his wife Dima for their great help and support before and after arriving to UBC, Omar Khasawneh, Mohammad Al-Rawashdeh and Mohammad Shamma for their friendship and advice, Elizeu Santos-Neto for the countless number of mind provoking debates and Lauro Costa for the interesting social and technical discussions.

My family has been the first and foremost source of support and inspiration. I am in eternal debt to my parents, Hassan and Amouneh, who always respect and support my choices, and encourage me to chase my dreams; no words can describe how grateful I am. My brother, Bashar, and my sisters, Sana’a, Muna, Salam and Suha, have always been my safety net on the tightrope of life, my deepest gratitude to them.
To my parents, brother and sisters
1. Introduction

Today’s large-scale applications (e.g., large-scale scientific applications, large-scale web services) demand not only high-speed computing power (e.g., supercomputers, cloud computing infrastructures), but also corresponding high-performance, efficient storage systems. As a result, efforts to improve data storage and retrieval resulted in a paradigm shift in storage systems design [1, 2]. So far, the development with the longest lasting impact appears to be the shift from centralized storage design to decentralized, distributed architectures.

On the one hand, this paradigm shift enabled several opportunities for storage systems to cope with the ever increasing demand from data intensive applications. Such opportunities include improved scalability, throughput, cost-effectiveness and reliability. On the other hand, designing distributed storage systems is a challenging task that requires trading-off one opportunity for another.

This thesis proposes and evaluates a novel distributed storage architecture that aims to maximize performance and reliability, while controlling cost.

The rest of this chapter discusses the motivation for this work (Section 1.1), outlines the opportunities and the solution (Section 1.2), presents the associated challenges (Section 1.3), describes the methodology (Section 1.4) and contributions (Section 1.5), and finally presents the structure of this thesis (Section 1.6).

1.1. Motivation

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 centralized
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 to increase performance, leads to increasingly costly systems.

A number of existing distributed storage systems (e.g., cluster-based [3-8] and peer-to-peer [9-13] storage systems) attempt to offer cost-effective, reliable data stores on top of unreliable, commodity or even donated storage components. To tolerate the failures of individual nodes, these systems use data redundancy through replication or erasure coding [14-16]. This approach, however, 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 increase of data volume stored when all other characteristics of the system are kept constant.

In summary, this thesis is motivated by the need for a high-performance, reliable and low-cost storage system that is able to cope with the requirements of data-intensive applications.

1.2. Opportunities and Solution

This study 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 [11]. 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 (e.g., ensuring four 9’s availability) 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 [17]. 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 [18] deployments are often supported by massive hardware resources (e.g., clusters and parallel file systems). Integrating 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. For instance, such storage service can be used to support photo-sharing web services (e.g., Flickr [19], Facebook [20]) which expose read-dominant workload, and require high level of reliability.

### 1.3. Challenges

The proposed hybrid architecture brings two challenges.

First, correctly provisioning 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 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, bandwidth, and disk space). Therefore, it is important 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, we aim to address the following more concrete questions:
Consider a system built atop of volatile, unreliable components only, and 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 replication and replica placement scheme enables maximum availability?*

### 1.4. Methodology

To answer the above questions, we present an analytical model based on Markov chains. The 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 used, and provides an accurate evaluation of the system performance.

Finally, to demonstrate feasibility and evaluate the performance characteristics of this architecture, we describe a proof of concept prototype based on the GridFTP usage scenario. The prototype integrates the Globus’ project GridFTP server [21] and MosaStore [8], a data store based on scavenged storage.

1.5. Contributions

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).
1.6. Thesis Structure

The rest of this thesis is organized as follows. Chapter 2 presents background and related work. Chapter 3 discusses our system model. Chapter 4 analyses the relationship between system’s characteristics and data availability through analytical modeling and simulations. Chapter 5 evaluates a use case application (a GridFTP server) for the proposed storage system. We conclude in Chapter 6.
2. Background and Related Work

This chapter presents background information related to redundant storage systems similar to our proposal (Section 2.1), and discusses efforts for analytical modeling of replicated storage systems (Section 2.2).

2.1. Redundant Storage Systems

A countless number of distributed storage systems use data redundancy to improve reliability. The differentiating factors that drive the design of these systems are related to the characteristics of the targeted deployment environment: the maintenance bandwidth available at the participating nodes as well as their failure and repair rates. Blake et al. [22] argue that bandwidth constraints ultimately determine the achievable storage capacity regardless of the redundancy level used; Chun et al. [23] also reached to the same conclusion. In these conditions, the following techniques have been widely used to reduce the volume of generated maintenance traffic:

- **Separating durability and availability**: Chun et al. [23] decouple durability and availability management to reduce maintenance costs in the presence of failures. Lefebvre et al. [24] 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, however, completely separates durability and availability management. Durability is related to the characteristics of the durable component; while the desired level of availability is controlled by correctly provision the system’s resources (e.g., the bandwidth of the durable and volatile nodes and the replication level). Hence,
allowing for 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 [23] 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. [22], 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.

- **Employing extra computing power.** Full replication and erasure coding are two widely employed data redundancy techniques. On the one side, full replication supports redundancy by creating several copies of the whole object. Full replication has the advantage of simplicity and low access overhead; however it imposes high bandwidth and storage maintenance cost.

  Erasure coding, on the other side, avoids whole object replication. It supports redundancy by splitting an object into \( m \) blocks and coding them into a larger set of \( n \) blocks, where an object can be reconstructed by decoding any \( m \) available blocks. Compared to full replication, erasure coding reduces storage maintenance cost; however it imposes extra computational costs due to coding/decoding.

  Our system employs redundancy at the volatile nodes to support better availability; the design, however, is agnostic to the redundancy mechanism in use as either erasure coding or full replication can be employed. In this work, however, we limit our
discussion on redundancy, and its effect on availability, to full replication while we leave erasure coding for future work.

The rest of this section classifies redundant storage systems into three high-level categories based on the targeted deployment environment. For each category, we first discuss examples while focusing on the mechanisms used to support data reliability, and then a summary and a comparison with our work.

2.1.1. Peer-to-peer (p2p) Storage Systems

Peer-to-peer storage systems harness idle disk space from thousands of wide-area, distributed workstations, and use redundancy to both reduce the probability of losing data and increase availability. Examples of such systems include OceanStore [9], TotalRecall [25], Fariste [11], CFS [10], PAST [26] and Ivy [27].

OceanStore [9], for example, aims to provide a global storage utility where customers pay to use the service. The system is designed with two primary goals in mind: first, the system is built atop of low-cost, untrusted, wide-area infrastructure: storage nodes may fail and lose data without notification; second, high reliability: since the system aims to offer a global storage utility, users must be able to access their data easily from any geographic location.

To support high reliability, OceanStore employs a two-tier architecture. The first tier, named the primary tier, consists of well-connected, high-bandwidth storage nodes. The primary tier acts as a distributed master replica of all objects, and as a frontend layer for clients to access their data, further it is responsible for ensuring data consistency. The second tier consists of a very large number of backend storage nodes responsible for reliably
storing the data. Reliability is ensured through erasure codes with high degree of redundancy.

Objects in OceanStore are modified by clients via updates, where each update creates a new version of the object. The system requires all updates to be processed first by the replicas in the primary tier, which runs a Byzantine agreement protocol to decide on the right commit order to ensure consistency. Once committed in the primary tier, updates are multicasted to the second tier using application-level dissemination trees. Moreover, as part of the update process, erasure codes are generated by the primary tier, and distributed widely in the second tier; providing, as a result, strong durability for the whole system.

In a sense, this hierarchical architecture decouples durability and availability: durability is maintained by employing erasure codes with high level of redundancy distributed over a second tier of a large number of globally distributed nodes. Availability, on the other side, is maintained by a primary tier of small number of high-bandwidth, high-connected nodes.

Ivy [27] and CFS [10] are other two P2P storage systems based on distributed hash table (DHT) schemes [28]. DHTs are distributed data structures that enable efficient lookup/store operations similar to typical hash tables. In a nutshell, the identifier space in a DHT is shared between the nodes and the objects, and is arranged in a ring. Based on this organization, an object is placed at its successor in the DHT ring, where a successor is the next node in the identifier ring when traversing the ring clockwise. Using DHTs as an underlying overly in distributed storage systems brings two advantages: first, it enables ro load-balance the objects across the participating storage nodes; second, it mitigates the need for maintaining explicit object-node mapping as this information is embedded in the object’s identifier, which is used as a key to lookup the object’s data.
To handle reliability, both Ivy and CFS use simple full replication technique: the $n$ replicas of an object are placed at the nodes immediately after an object’s successor in the DHT ring. The assumption is that the nodes close to each other in the DHT ring are not likely to be close physically, hence lowering the probability of correlated failures that could hit all the replicas of a specific object.

**Summary:** Managing data reliability in wide-area, P2P storage systems is a challenging task. This is due to bandwidth limitations inherent in wide-area systems and to their dynamic characteristics (e.g., the availability of participating nodes depends on factors that include users’ behavior). As a result, to efficiently support reliability, different P2P storage systems employ different techniques to improve reliability depending on the targeted workloads.

Our design, however, targets a different deployment environment: the system is built atop of LAN connected components; hence the participating hosts enjoy an infrastructure with better connectivity and individual nodes’ availability compared to wide-area storage systems.

### 2.1.2. Cluster-based Storage Systems

Cluster-based storage systems are hosted on dedicated, well-provisioned infrastructure, rendering reliability management a simpler task compared to P2P storage systems. Examples of such systems include GFS [3], Ceph [5] and PVFS [29].

*Google file system* (GoogleFS) [3], for instance, is built atop of a large number of commodity, yet dedicated hosts. The system employs simple full replication technique to support reliability; specifically, it uses a default replication level of three. To avoid the development of hot spots, the replication level is raised for objects that are accessed
frequently, improving their availability as a result. Likewise, Ceph [5] employs basic $n$-way replication to mask storage device failures.

Surprisingly, PVFS [29] does not explicitly address the problem of reliability. The assumption is that reliability is handled at the disk layer by employing RAID-like techniques.

**Summary:** Cluster-based distributed storage systems operate in a different deployment infrastructure that includes dedicated, trusted, highly-available storage nodes that are connected via high-capacity network. The goal of these storage systems is to achieve maximum throughput; hence, reliability is managed via simple replication techniques that do not compromise access performance.

Similar to our argument in the previous section, compared to cluster-based storage systems, we target a different deployment environment: the participating hosts include a mix of dedicated storage nodes (the durable components) and unreliable storage nodes (the volatile components). As a result, our architecture aims to offer the high-performance characteristics of the cluster-based storage, while reducing deployment costs.

### 2.1.3. Scavenged Storage Systems

Closer to our system are *scavenged storage systems* (Freeloader [4], dCache [30], MosaStore [8]) which aggregate idle storage space from LAN-connected non-dedicated workstations.

*Freeloader* [4], for example, aims to build a site-local cache space to improve access performance of large datasets in data-intensive scientific computing. The system assumes that datasets’ reliability is maintained by a primary storage system, and Freeloader is used as a high-performance scratch space to feed the computations. The feasibility of this model
stems from the fact that datasets in scientific applications have mostly write-once-read-many access pattern.

*dCache* [30], however, combines heterogeneous disk-based storage systems (including donated storage) to offer a unified, high-performance, large-scale data store. Similar to our proposed architecture, *dCache* enables the inclusion of a tertiary storage system as a back-end reliable data store. Further, *dCache* allows for replicating files to improve reliability; *dCache* designers, however, do not provide guidance on how to provision the system to provide user-defined guarantees for data availability and durability.

**Summary:** Scavenged storage systems aim to build high-performance storage space similar to that offered by cluster-based systems, while reducing the associated costs (in terms of dollar per storage space unit). Our system has the same goal, yet, at the same time aims to include an additional property: strong reliability. Further, we provide tools that assist in provisioning the system based on the deployment requirements.

### 2.2. Analytical Models of Replicated Storage Systems

Markov chains have been used in the past to model replication-based systems. Chun et al. [23] study durability in large scale distributed storage systems built atop of nodes spread over the Internet. The study assumes a symmetric system in which storage nodes share similar characteristics. The study employs Markov chains to derive an expression for the number of expected replicas of an object for a given replica creation and failure rates. A similar model is also considered in [31, 32].

Ramabhardan et al. [33] address the problem of how to provision a peer-to-peer storage system that employs replication to reliably store data beyond the lifetime of the individual storage nodes. The study is based on a simple Markov model that captures the fundamental
characteristics of a replicated system (e.g., replica creation and failure rates). The model is then analyzed to derive an expression for the expected object lifetime, and to examine the impact of resource constrains (e.g., storage and repair bandwidth limitations) on durability, and how to optimally tune these constrains to maximize durability.

Lian et al. [34] considers Markov chain model to analyze the impact of replica placement on the durability of cluster-based distributed storage systems. The model is used to derive an expression for mean-time to data loss (MTTDL), which measures the average time the system can hold before it permanently loses the first data object.

Our analysis is different in that it considers an asymmetric system. We base our model on the one proposed by Chun et al. [23]. We expand the model to add the durable component, and derive an expression for object availability.
3. System Model

This chapter describes the main components of the proposed storage architecture (Section 3.1), details the assumptions our analysis is based on (Section 3.2) and discusses a number of questions related to the model (Section 3.3).

3.1. Components

Our architecture is composed of three main components (Figure 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 frontend storage area with substantial parallel IO channels, hence supporting high-performance data access.

- One durable component. In reality, the durable component could be an external archival service or a complex set of devices (e.g., automated tape library) but, for the purpose of our
analysis, we model it as a single durable component that maintains a copy of all objects in the system. This forms a backend data store that recreates the data the volatile nodes may lose; offering, accordingly, strong durability guarantees for the whole system.

- **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 more resilient to failures, are possible [5, 35].

### 3.2. Assumptions

- **Volatile nodes’ failure model:** 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. Failures are detected using timeouts. Volatile nodes declare their availability to the system using heart-beat messages sent to the metadata service.

- **Service model:** The clients access the system only through the volatile nodes (Figure 1). 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. Note that clients never read or write data through the metadata service, hence keeping it outside the critical I/O path.

- **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 is allowed a limited amount of repair bandwidth \( b \). If the replication level on the volatile nodes goes down to 0 for a specific object, 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 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 on the durable component.

Regardless of the approach used to handle writes, our study demonstrates 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.
2.) Can workload characteristics be exploited to maximize availability, and equivalently reduce maintenance costs?

Workload characteristics can be used to improve availability and/or reduce maintenance costs. For example, ideas similar to the ones presented in TotalRecall [25] can be applied to further reduce maintenance costs and improve objects availability. TotalRecall aims to dynamically adjust the redundancy configuration of individual objects to achieve a particular availability requirement, while minimizing the associated redundancy costs. Specifically, the system automatically adjusts the redundancy mechanism (erasure coding vs. full replication) and the redundancy level according to real-time availability measurements of storage nodes as well as workload characteristics. For instance, full replication has high storage overhead, but low run-time access latency (no processing required to access the object), therefore it is suited for random access patterns and for small-sized files; erasure coding, however, has much less storage requirements compared to pure replication, but suffers high run-time access latency (due to coding/encoding); therefore using erasure coding for large-sized files, for example, would be space efficient.

Another idea is to prioritize the repair of frequently accessed objects to improve availability. Prioritization can be done at the durable node when repairing a frequently used lost object, as well as at the volatile nodes.

3.) Where do high-level file system properties fit in the proposed architecture?

This work proposes a storage system architecture, not a file system. Therefore, high-level file system properties, such as security and consistency semantics, are outside the scope of our study. However, the characteristics of the architecture (e.g., the fact that it is distributed)
may impact the design of such features. For example, security issues related to distributed storage have been addressed extensively in the literature [36-40].

4.) *What is the effect of data striping on availability?*

As we discussed in the previous chapter, parallel file systems typically employ striping to improve performance. Striping, however, reduces availability as each data segment must be available for the entire file to be available. In chapter 4 we study object’s availability without considering the effect of striping. However, to calculate the availability of a striped object, each data segment can be considered as a different non-striped object; therefore, analytically, if an object is divided into *c* segments, then its availability is $A^c$, where $A$ is the availability of a single segment.
4. Availability Study

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 (the second assumption in the previous section). Therefore, in this context, we define availability as the ratio of the total time an object replica exists on at least one volatile node.

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

- **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 (Section 4.1.1), derives an expression for object availability (Section 4.1.2), and discusses the accuracy of the model (Section 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 2: Markov chain model. Each state represents the number of currently available replicas at the volatile nodes. There is a transition out of state 0 with a special rate $\lambda_0$ which depends on the durable nodes’ bandwidth.

Figure 2 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 $\lambda_k$ corresponding to the replica repair rate. For $k = 0$, the repair rate $\lambda_0$ 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 $\lambda$. This repair rate $\lambda$ 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 $\mu$ 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 $\mu$ and the repair rates $\lambda_k$ 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 $n$). 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 [41]:

\[
p_0 = \frac{1}{1 + \sum_{k=1}^{n} \prod_{i=0}^{k-1} \frac{\lambda_i}{\mu_i}}
\]

However,

\[
\lambda_k = \begin{cases} 
\lambda_0 & k = 0 \\
\frac{\lambda}{\mu} & 0 < k \leq n \\
0 & k > n
\end{cases}
\]

\[
\mu_k = k\mu \quad k \geq 0
\]

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

\[
p_0 = \frac{1}{1 + \sum_{k=1}^{n} \frac{\lambda_0(\frac{\lambda}{\mu})^{k-1}}{k!}} \frac{1}{k!}
\]

Two notations better reveal the meaning of the above formula. We refer to the term \( \frac{\lambda}{\mu} \) as \( \rho \); intuitively \( \rho \) 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 life time of a volatile node). Additionally we will refer to the term \( \frac{\lambda_0}{\mu} \) as \( \gamma \), 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 $\rho$ (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 $\rho^{k-1}$ for $n > \rho$. Second, if $\rho$ is large enough, then a reasonable value of $\gamma$ is enough to achieve good availability (i.e., low $p_0$).

Table 1 presents the probability for object unavailability ($p_0$) for replication levels often used in practice. Note that when $n = 1$, availability depends only on the durable node repair ratio $\gamma$, which is intuitive. Also, as expected, there are decreasing incremental gains from increasing the replication level.

<table>
<thead>
<tr>
<th>Replication level ($n$)</th>
<th>Probability of state $p_0$ (un-availability)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$1/(1 + \gamma)$</td>
</tr>
<tr>
<td>2</td>
<td>$1/(1 + \gamma(1 + \frac{\rho}{2}))$</td>
</tr>
<tr>
<td>3</td>
<td>$1/(1 + \gamma(1 + \frac{\rho}{2} + \frac{\rho^2}{3!}))$</td>
</tr>
</tbody>
</table>

4.1.3. Model Accuracy

This section summarizes the limitations of the analytical model presented above.

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 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 [32, 33, 42]; 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 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 $\lambda_k$ will depend on the number of replica sources $k$ and the number of replica destinations $n - k$, and it is expressed as $\lambda \cdot \min(n - k, k)$. However, to keep the analytical model tractable, $\lambda_k$ 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
course-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

To overcome the limitations of the analytical model discussed in the previous section, we
use SimPy [43], a process-based discrete-event simulation language, to build a simulator
that provides a more accurate view of the proposed system. The reset of this section
discusses the simulation model (Section 4.2.1), the accuracy of the model (Section 4.2.2)
and how we validate the simulator (Section 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 will be 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). Hence a node’s upload channel is not time-shared and is dedicated for one transfer at a time. 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 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

Table 2 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, network contention is modeled only on the access link to each node; 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; further, our experiments in the evaluation section, which measures the peak bandwidth during the simulations, supports the adequacy of this assumption.
Table 2: A summary of factors that affect system state and the way they are captured by the analytical and simulation models.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Analytical model</th>
<th>Simulation model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transient failures.</td>
<td>Not modeled</td>
<td>Trace-driven.</td>
</tr>
<tr>
<td>Permanent failures.</td>
<td>Exponentially distributed failure time.</td>
<td>Trace-driven.</td>
</tr>
<tr>
<td>Number of objects and their placement on the volatile nodes.</td>
<td>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.</td>
<td>Replicas are distributed according to a specific placement policy which is accurately simulated.</td>
</tr>
<tr>
<td>Replica repair rate (in case more than one replica is needed to obtain the target replication level $n$ for a specific object).</td>
<td>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.</td>
<td>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.</td>
</tr>
<tr>
<td>Bandwidth limits for replica repair at the durable and volatile nodes.</td>
<td>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.</td>
<td>Fixed maximum rate. The simulator also models contention on a node’s access link by queuing replica creation operations.</td>
</tr>
<tr>
<td>Replica management overhead (e.g., time to detect a failure, decide where to get a replica from and where to place it).</td>
<td>Not part of the model.</td>
<td>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.</td>
</tr>
<tr>
<td>Correlated replica failures.</td>
<td>Replicas are assumed to fail independently; however, in reality, multiple replicas exist on the same node, hence failures are correlated.</td>
<td>Exact placement is modeled with multiple replicas of different objects on the same node, hence replicas may fail simultaneously.</td>
</tr>
</tbody>
</table>

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. [44] demonstrated that Weibull distributions model well desktop availability in
an enterprise environment, Nurmi et al. [45] also reached to the same conclusion. On the other side, exponential distributions are widely accepted to model well permanent failure inter-arrival times [23, 31-33].

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 [44-46], 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 (i) gain additional insight beyond what the analytical model can offer; (ii) compare the characteristics of our design with those of systems built atop of unreliable components only, and (iii) evaluate the effect of the 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 1GB objects. The replication level $n$ is set to 3, a common value used in distributed storage systems [3].

The impact of object size can be summarized as follows. On the one side, using very small object sizes increases the opportunity for parallel repair, however it produces large number of objects that exhaust 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 less number of 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 [26].

The system contains 1000 volatile nodes, each allocates 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 [47].

Further, we use 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 [44], and the corresponding
node MTTF is 20 days). For the permanent failures, the exponential distribution has a node MTTF of one year (similar to [23]).

Finally, we report unavailability rather than availability as it is easier to plot on logarithmic scale. For all experiments, we report averages over at least 30 runs 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 3 demonstrates the result.

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 \( \rho \) and \( \gamma \). To do so, we need to estimate the failure rate \( \mu \), and the creation rates \( \lambda \) and \( \lambda_0 \) implied in the traces used.

On the one hand, the replica failure rate \( \mu \) 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 \( \mu \) 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 \( \lambda \) 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 $\lambda$ of a single object can be estimated by $2*b/d$.

![Graph](image)

**Figure 3: Comparing unavailability predicted by the analytical model and simulations (the lower the better).**

In this experiment, there are 1000 volatile nodes, and the average amount of data maintained by each node is $d = 32*3/1000TB \approx 98GB$; thus, for example, for the first data point where $b = 1Mbps$, $\lambda \approx 78$ replica copies per year. The same estimation also applies to $\lambda_0$ by replacing $b$ with the durable node’s repair bandwidth.

These estimates for the creation rates $\lambda_0$ and $\lambda$ are conservative as they assume that a failed node’s data is 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, Figure 3 shows the analytical model predicting conservative availability (i.e., higher unavailability) compared to simulations.

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

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 are interested in answering the following question: *What is the maximum amount of data that can be preserved durably during a specific time interval given particular system characteristics* (e.g., repair bandwidth and replication level)?

![Figure 4: The fraction of objects lost after four years. For storage load less than 16TB, no objects were lost in any configuration. When aggressively increasing the replication level and the bandwidth limits \( n = 8 \) and \( b = 8 \text{Mbps} \), the system does not lose any objects for storage load up to and including 128TB.](image-url)
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 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 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 $n = 8$ and $b = 8$Mbps, the system does not lose any objects while maintaining up to 128TB storage load. However, increased durability comes at the cost of increased maintenance traffic and disk space. Further, when the storage load increases beyond 128TB, 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 terms of 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: $B = 1$Mbps, $b = 2$Mbps and $n = 3$ 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 128TB: $b = 8$Mbps and $n = 8$ replicas.
Figure 5: Total traffic sent at end of 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, $n = 8$.

Figure 5 demonstrates that the system with the durable component generates 60% less maintenance traffic than the system without durable component. This also corresponds to 60% lower average bandwidth consumed over the 4 year simulation period. For example, for storage load of 128TB, the system without durable component generates on average an aggregate traffic of about 343Mbps, a significant burden even for a gigabit LAN compared to only 133Mbps for the system with durable component.

To further investigate the behavior of the replication bandwidth, Figure 6 compares the cumulative distribution function (CDF) of aggregate bandwidth consumed for storage load of 128TB, while Table 3 summarizes the statistics of the distribution for the same experiment.

The results show that 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 reduction in maintenance cost. Moreover, the system with durable component experiences much lower maintenance traffic volume variability than the
system without durable component, making the replica maintenance overhead of the former
more predictable; reducing, as a result, the impact on application traffic.

Table 3: Statistics of aggregate bandwidth consumed for storage load of 128TB

<table>
<thead>
<tr>
<th>Configuration</th>
<th>With durable component</th>
<th>Without durable component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>133</td>
<td>343</td>
</tr>
<tr>
<td>Median</td>
<td>122</td>
<td>280</td>
</tr>
<tr>
<td>90(^{th}) percentile</td>
<td>214</td>
<td>560</td>
</tr>
<tr>
<td>99(^{th}) percentile</td>
<td>312</td>
<td>880</td>
</tr>
<tr>
<td>Maximum</td>
<td>892</td>
<td>6,472</td>
</tr>
</tbody>
</table>

Figure 6: CDF (up to 99\(^{th}\) percentile) of aggregate replication bandwidth consumed for
storage load of 128TB.

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.

The price paid for these savings is twofold: a slightly more complex, asymmetric
system that includes the durable component and lower availability. Figure 7 plots the CDF
of object unavailability for the system with durable component, while Table 4 summarizes
the statistics of the distribution.
Although unavailability increases as storage load increases, the system is still able to provide acceptable average availability levels even for large storage volumes, note that a significant percentage of the objects (90%) have about the same availability level as the average. Further, a significant percentage of the objects have full availability (97% for the smallest storage load used in experiment 16TB, and 60% for the largest storage load used of 128TB).

Table 4: Statistics of unavailability (the lower 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 $n = 3$.

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

Figure 7: CDF of unavailability (the lower 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 $n = 3$.

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

4.3.4. Provisioning System Resources

The next questions we consider are: *What is the effect of bandwidth limits at the durable and the volatile nodes on the achieved availability and the generated network traffic?* And, similarly, *What is the effect of the replication level on the achieved availability and the generated network traffic?*

### 4.3.4.1 Impact of bandwidth limitations at the durable node on availability and volume of generated traffic

Figure 8 plots the CDF of object unavailability while increasing the durable node’s bandwidth (Table 5 summarizes the statistics of the distribution). The results show that increasing the bandwidth of the durable component does not dramatically enhance availability. For example, in this experiment, reducing average unavailability by one order of magnitude requires increasing the durable component’s bandwidth by more than 8 fold while keeping the volatile node’s bandwidth constant.

<table>
<thead>
<tr>
<th>Durable node’s bandwidth ($B$)</th>
<th>1 Mbps</th>
<th>2 Mbps</th>
<th>4 Mbps</th>
<th>8 Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>1.90*10^{-5}</td>
<td>9.78*10^{-6}</td>
<td>7.09*10^{-6}</td>
<td>4.54*10^{-6}</td>
</tr>
<tr>
<td>Median</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>90th percentile</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>99th percentile</td>
<td>5.17*10^{-4}</td>
<td>4.52*10^{-4}</td>
<td>3.69*10^{-4}</td>
<td>3.44*10^{-4}</td>
</tr>
<tr>
<td>Maximum</td>
<td>4.93*10^{-3}</td>
<td>2.95*10^{-3}</td>
<td>1.15*10^{-3}</td>
<td>1.07*10^{-3}</td>
</tr>
</tbody>
</table>
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 the next section).

Figure 9 and Table 6 illustrate the corresponding aggregate bandwidth consumed. Increasing the stable component bandwidth does not affect the generated maintenance traffic. However, the slightly better availability is achieved at the cost of enabling faster recovery when a burst of failures hit all the replicas of a group of objects.

Table 6: Statistics of total aggregate replication bandwidth consumed at the durable node (Mbps) while varying the durable node’s bandwidth ($B$).

<table>
<thead>
<tr>
<th>Durable node’s bandwidth ($B$)</th>
<th>1 Mbps</th>
<th>2 Mbps</th>
<th>4 Mbps</th>
<th>8 Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>41</td>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>Median</td>
<td>24</td>
<td>26</td>
<td>28</td>
<td>32</td>
</tr>
<tr>
<td>90th percentile</td>
<td>94</td>
<td>96</td>
<td>98</td>
<td>104</td>
</tr>
<tr>
<td>99th percentile</td>
<td>196</td>
<td>194</td>
<td>198</td>
<td>202</td>
</tr>
<tr>
<td>Maximum</td>
<td>892</td>
<td>906</td>
<td>906</td>
<td>920</td>
</tr>
</tbody>
</table>
This is an important result if we consider the durable component as an outsourced storage utility. For example, if Amazon S3 offers a ‘premium’ service with better access performance, the system should not buy it as availability will not proportionally be enhanced.

4.3.4.2 Impact of bandwidth limitations at the volatile nodes on availability and volume of generated traffic

Figure 10 and Table 7 report unavailability while varying the volatile nodes’ bandwidth. Unlike the effect of the durable node’s bandwidth, unavailability decreases prominently as the volatile node’s bandwidth increases. For example, reducing average unavailability by one order of magnitude requires increasing the volatile nodes’ bandwidth 4 fold 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.
Table 7: Statistics of unavailability (the lower the better) while varying the volatile nodes’ bandwidth (b).

<table>
<thead>
<tr>
<th>Volatile nodes’ bandwidth (b)</th>
<th>1 Mbps</th>
<th>2 Mbps</th>
<th>4 Mbps</th>
<th>8 Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>$7.07 \times 10^{-5}$</td>
<td>$1.97 \times 10^{-5}$</td>
<td>$7.05 \times 10^{-6}$</td>
<td>$3.44 \times 10^{-6}$</td>
</tr>
<tr>
<td>Median</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>90th percentile</td>
<td>$1.90 \times 10^{-4}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>99th percentile</td>
<td>$1.18 \times 10^{-3}$</td>
<td>$5.17 \times 10^{-4}$</td>
<td>$2.86 \times 10^{-4}$</td>
<td>$7.93 \times 10^{-5}$</td>
</tr>
<tr>
<td>Maximum</td>
<td>$6.07 \times 10^{-3}$</td>
<td>$4.93 \times 10^{-3}$</td>
<td>$4.03 \times 10^{-3}$</td>
<td>$4.01 \times 10^{-3}$</td>
</tr>
</tbody>
</table>

Figure 10: CDF of unavailability (the lower the better) while varying the volatile nodes’ bandwidth (b). Note that the Y-axis starts at 80%.

Interestingly, the total amount of traffic generated (represented by the average bandwidth over the simulation period) does not increase proportionally with the significant observed improvement in availability.

Table 8: Statistics of aggregate replication bandwidth consumed (Mbps) while varying the volatile nodes’ bandwidth (b).

<table>
<thead>
<tr>
<th>Volatile nodes’ bandwidth (b)</th>
<th>1 Mbps</th>
<th>2 Mbps</th>
<th>4 Mbps</th>
<th>8 Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>38</td>
<td>40</td>
<td>41</td>
<td>42</td>
</tr>
<tr>
<td>Median</td>
<td>31</td>
<td>24</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>90th percentile</td>
<td>71</td>
<td>94</td>
<td>116</td>
<td>136</td>
</tr>
<tr>
<td>99th percentile</td>
<td>120</td>
<td>196</td>
<td>292</td>
<td>424</td>
</tr>
<tr>
<td>Maximum</td>
<td>438</td>
<td>892</td>
<td>1,864</td>
<td>3,616</td>
</tr>
</tbody>
</table>
The improvement in availability is achieved at the cost of allowing larger traffic spikes, which help to contain failure bursts. This is demonstrated by the increase in the length of the right tail of the traffic distribution.

Figure 11: CDF of total aggregate replication bandwidth consumed (Mbps) while varying the volatile nodes’ bandwidth (b).

4.3.4.3 Impact of replication level

Figure 12 plots the CDF of object unavailability while varying the replication level (Table 9 reports the summary statistics of the distribution). As expected, increasing the replication level decreases unavailability. The interesting result is that for each additional replica, average unavailability decreases by one order of magnitude. This is possible because the system is configured with enough bandwidth to support the examined replication levels.

Table 9: Statistics of unavailability (the lower the better) while varying the replication level.

<table>
<thead>
<tr>
<th>Replication level (n)</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>1.97*10^{-5}</td>
<td>1.49*10^{-6}</td>
<td>1.39*10^{-7}</td>
<td>2.46*10^{-8}</td>
</tr>
<tr>
<td>Median</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>90th percentile</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>99th percentile</td>
<td>5.17*10^{-4}</td>
<td>5.70*10^{-6}</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Maximum</td>
<td>4.93*10^{-3}</td>
<td>3.99*10^{-3}</td>
<td>3.23*10^{-4}</td>
<td>2.42*10^{-4}</td>
</tr>
</tbody>
</table>
Figure 12: CDF of object unavailability (the lower the better) while varying the replication level $n$.

Note that for all replication levels at least 92% of the objects are fully available, further, and equally important, the maximum unavailability improves proportionally as well.

Figure 13 reports the corresponding replication bandwidth distribution (Table 10 summarizes the statistics). The significant gain achieved in availability when increasing the replication level comes at a modest increase in replication bandwidth cost in terms of average as well as peak bandwidth: 29% increase in the peak bandwidth when increasing the replication level from 3 to 4, down to only 10% when increasing the replication level from 5 to 6. Still, increasing the replication level incur additional costs in terms of storage space.

Table 10: Statistics of aggregate replication bandwidth consumed (Mbps) while varying the replication level. The system is configured with $B = 1$Mbps, $b = 2$Mbps and for storage load of 32TB.

<table>
<thead>
<tr>
<th>Replication level ($n$)</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
</tr>
<tr>
<td>Median</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
</tr>
<tr>
<td>90th percentile</td>
<td>94</td>
<td>120</td>
<td>138</td>
<td>158</td>
</tr>
<tr>
<td>99th percentile</td>
<td>196</td>
<td>244</td>
<td>286</td>
<td>336</td>
</tr>
<tr>
<td>Maximum</td>
<td>892</td>
<td>1152</td>
<td>1322</td>
<td>1458</td>
</tr>
</tbody>
</table>
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 14: Comparing average unavailability for three different replica placement schemes.

Figure 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 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.
5. **Use Case: A GridFTP Server**

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

GridFTP [18] 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.

We build a GridFTP server that employs the proposed hybrid architecture. Our GridFTP server is based on integrating components from two systems (Figure 15): (i) a scavenged storage system, named MosaStore [8], and (ii) Globus’ project GridFTP server [18].

MosaStore’s design consists of three main components: a metadata service, a set of donated nodes (named benefactors) and the clients that access the system. Globus’ GridFTP server, on the other side, consists of a single frontend protocol interpreter node that parses GridFTP clients’ commands, and a set of storage nodes that run the data transfer processes (DTPs) which are responsible for data movement.
Figure 15: High-level architecture of the GridFTP use case. The server is built using two components: Globus’ GridFTP Server and MosaStore scavenged storage system.

Using components from the above-described systems, we build our GridFTP server. MosaStore’s metadata service resembles the centralized metadata service in our proposed hybrid architecture, the benefactors represent the volatile nodes in the system; moreover, the benefactors run the GridFTP’s DTP component which handles data access requests from the GridFTP protocol interpreter. Finally, the newly integrated server includes an additional component to represent the durable node.
The main challenge in this integrated architecture relates to the transparent integration of these components: users should not perceive any difference between the service provided by the new GridFTP server and a standard GridFTP server.

The rest of the chapter presents background related to GridFTP protocol and standard GridFTP server deployments (Section 5.1), discusses limitations of current GridFTP deployments and presents potential solutions (Section 5.2), describes in details our GridFTP design that employs the proposed hybrid architecture (Section 5.3) and a brief evaluation of a prototype implementation (Section 5.4).

5.1. Background

GridFTP [18] has become the data access protocol of choice for data-intensive scientific communities. GridFTP protocol is based on the highly-popular File Transfer Protocol (FTP) [48] with the addition of new features that support the requirements of data grid applications. One important feature is the support of parallel transfers, in which data distributed across a set of hosts at one end is transferred using multiple, independent data streams to another remote set of hosts to take advantage of the independent I/O paths between the participating hosts.

The rest of this section first presents an overview of the relevant basic FTP operations (Section 5.1.1), next we describe how the GridFTP protocol extends FTP operations to support parallel transfers (Section 5.1.2) and finally discusses standard GridFTP server deployments (Section 5.1.3).
5.1.1. Typical FTP Operations

An FTP server listens for clients’ connections on a designated TCP port (default 21). A client connects to this port to establish a control channel through which the client sends commands to the server, and the server sends back responses. The actual data transfer occurs via a second channel, named the data channel. The data channel can be established in two ways: in passive mode, the client sends the server a dynamic address (IP and port number), and the server connects to this address to start the data transfer; in active mode, the server sends the client a dynamic address, and hence the client establishes the connection.

To facilitate this separation between control and data channels, a standard FTP server design consists of two main components: (i) Protocol Interpreter (PI) that serves as a frontend handler to the commands sent by the clients over the control channel and (ii) Data Transfer Process (DTP) that is responsible for access to storage and data transfer over the data channel.

This decoupling of control and data channels enables the establishment of third-party transfers: a transfer between two servers mediated by a client (Figure 16 (left)). Particularly, in a third-party transfer, the client sends an FTP read command in passive mode to the source server. The PI at the source server interprets the command, and as a result runs a DTP instance. As a response to the passive read command, the PI passes to the client, through the control channel, the address the new DTP is listening on. Next, the client sends a write command in active mode to the destination server. As a parameter to the active write command, the client passes the address obtained previously from the source server. The PI at the destination server runs a DTP process and delegates to it the write command along
with the source DTP address. Finally, the destination DTP establishes a data channel with the source DTP, and finally pulls the data directly.

![Diagram of FTP and GridFTP server designs](image)

**Figure 16**: A typical FTP server design (left), Globus's GridFTP server design in a typical deployment (right) in the context of a third-party transfer.

### 5.1.2. GridFTP Enhancements Over FTP

In the previously discussed FTP protocol, only one data channel is established per file transfer. A parallel data transfer is facilitated by allowing for more than one DTP to collaborate in a transfer. Therefore, the GridFTP protocol extends the FTP protocol to enable passing more than one DTP address per transfer; further, it extends the data channel protocol to enable splitting files into smaller blocks among participating DTPs.

Figure 16 (right) demonstrates Globus’s GridFTP server design [21], a widely used GridFTP server implementation. To further support parallel data transfers, the design completely separates the PI and the DTP modules (they communicate through an IPC channel), thus enabling the deployment of DTP modules on different hosts, and independently from the PI.
5.1.3. **Standard GridFTP Deployment**

A typical GridFTP server setup uses at least one PI server running on a designated node known by the clients and one DTP process on each host responsible for data transfers. Note that to support parallel transfers, all DTPs involved in the transfer should have access to the file to be transferred, where each DTP is responsible for a specific part of the file. Additionally, current DTP implementations access the underlying file system by issuing standard POSIX calls; hence a shared file system that exposes standard POSIX interface must be used to support this configuration.

5.2. **Standard GridFTP Deployments: Limitations and Solutions**

This section discusses limitations, and corresponding solutions, associated with standard GridFTP deployments: relatively high deployments costs (Section 5.2.1) and relatively low performance efficiency (Section 5.2.2).

5.2.1. **Reducing Deployment Costs**

To offer high-performance, GridFTP deployments are customarily supported by high-end, expensive hardware resources that enable high I/O access rates, such as parallel file systems (e.g., PVFS [29], HPSS [49]) over dedicated clusters. This is often not affordable or justifiable in research environments, hence it increases the barrier for sites to join the grid, collaborate, and share data.

To reduce the cost of GridFTP deployments while maintaining their high-performance and reliability characteristics, we propose to build a GridFTP server atop the hybrid storage architecture which combines a low-cost durable component and volatile storage.
On the cost side, the durable component can be represented by a low-cost Automated Tape Library (ATL), while the volatile storage is obtained by harnessing the free, underused storage capacities that are already available on the site’s desktops, and are already maintained by site administrators.

In this context, data durability is preserved by the backend durable component, while data availability can be tuned based on site-specific requirements (chapter 4 discussed in details the trade-offs in balancing availability and maintenance overhead).

Finally, on the performance and scalability side, scavenged storage inherits the performance and scalability characteristics of cluster-based storage: desktops in sites are usually connected via high-bandwidth local area network; moreover, as the site grows, more desktops participate in the system, inherently increasing the I/O bandwidth and storage space of the system. However, an innovative GridFTP design is still required to maximize the utilization of this high-potential environment, which we discuss in the next section.

5.2.2. Unleashing the Maximum Performance of GridFTP Deployments

The second limitation of standard GridFTP deployments is related to the low efficiency achieved via the previously-described standard setup. In practice, parallel file systems use data striping to support high-performance data transfers: a file is divided into segments that are distributed across multiple storage nodes (SNs). This technique enhances scalability and offers significant performance improvements as it assembles a ‘fatter’ I/O channel from multiple storage nodes. At the same time, parallel file systems typically include a client module that allows upper-level applications to seamlessly access the system using regular POSIX I/O functions. Specifically, the client module accesses a file by aggregating the file’s
data segments in parallel from the storage nodes, practically aggregating the bandwidth of multiple disks.

In a GridFTP deployment over a parallel file system, each DTP accesses the parallel file system through the above-described client module by issuing standard POSIX I/O calls (Figure 16 (right)).

Although relying on the standard POSIX interface makes the DTPs completely agnostic to the details of the underlying parallel file system, it severely limits the GridFTP server performance: each DTP is responsible for a portion of the file, hence each DTP transfers its portion in two stages, first the client module pulls the data from the underlying storage nodes as a response to the DTP request, second the DTP transfers the data to the corresponding destination DTP. As a result, the client module introduces an extra layer that (i) hides data distribution details, hence not taking advantage of data locality and (ii) introduces added overhead in terms of extra data transfers and system calls at the source and destination DTPs, the main reasons for limited throughput.

To alleviate this limitation we propose to deploy the DTPs directly on the storage nodes (the volatile nodes in the hybrid architecture). This solution effectively enables exploiting data locality, and increases the granularity of parallelism. However, this requires two major modifications to GridFTP server design.

First, the PI module must be aware of the details of data segments’ placement across storage nodes. This is required because, in this setup, each DTP has access to only the data stored on the local storage; therefore, for each file transfer, the PI has to forward transfer requests to only the DTPs that have access to the file’s data segments. Second, the DTP
module must be aware of what data segments are stored on the local storage, and how to access them.

The disadvantage of these modifications is that they require file system specific information, such as the location of data segments of the file to be transferred, how these segments are stored on the storage nodes and how they can be accessed.

Although these modifications make the GridFTP server tightly coupled with the underlying file system in use, they still can be transparently integrated with respect to the GridFTP client and other GridFTP servers in case of third-party transfers.

5.3. A Scavenged GridFTP Server

This section presents the design of a GridFTP server that puts the above-discussed solutions in practice. We describe in details the building components (Section 5.3.1) and illustrate how these components interact in practice (Section 5.3.2).

5.3.1. Server Design and Components

As we mentioned at the beginning of this chapter, our GridFTP server design integrates components from two systems (Figure 17):

First, Globus’ project GridFTP framework [21]: 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 by the underlying storage system. First, the control interface integrated with the PI module to form the GridFTP PI. The PI module parses GridFTP commands received on the control channel, and invokes the appropriate handlers from the control interface which, in turn, carries on storage
system specific handling. Second, the data interface controls the Data Transfer Process (DTP), which handles access to stored data and its transfer through the data channel.

Second, MosaStore [8]: a highly configurable scavenged storage system designed to harness unused storage space from LAN-connected workstations. MosaStore consists of two main components: a metadata service and the benefactor daemons which run on the nodes that donate storage space. Similar to the standard practice in parallel file system design, MosaStore employs striping to increase performance: files are divided into smaller chunks distributed among the benefactors.

Figure 17: GridFTP/MosaStore integrated architecture. The figure demonstrates a third-party transfer: the client communicates with the Server PI of both servers using standard GridFTP commands. The Server PI, in its turn, contacts the responsible DTPs (running on the volatile nodes) using an internal IPC channel to start the transfer.

The metadata service maintains information related to the available space, the 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-/off-line, 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.

The main challenge relates to the transparent integration of these components. Figure 17 presents the integrated architecture and outlines the new components we added. A GridFTP/MosaStore integrated server consists of the following components:

1.) **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.

2.) **The DTPs** run on the donated storage nodes (representing the volatile nodes in the hybrid architecture). Each DTP is composed of three components: (i) MosaStore’s benefactor daemon which manages local storage and replication requests from the metadata service, (ii) the data channel controller implements the standard GridFTP data channel protocol and handles data movement from/to the node and (iii) the data interface which handles data access requests from the server PI by conveying data between the benefactor daemon and the data channel controller.

3.) **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.

4.) **The Durable Component Daemon** operates on the durable component where it handles replication requests. Note that durable component does not run a DTP as it has no rule in serving data to clients.
5.3.2. Example of Component Interaction

In the section 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. The main role of the client is to relay DTP addresses between the source and destination GridFTP servers.

Specifically, a third-party transfer is accomplished as follows:

1.) 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 its turn, contacts the metadata service asking for the chunk-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.

2.) 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.

3.) 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.

4.) Data Movement. At this point, the control interface at the destination server has identified the source and destination DTPs. Subsequently, the control interface delegates
5.) Replication Management. 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.

5.4. 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. For all experiments we report averages over 30 runs
with 95% confidence interval. We compare our server with unmodified GridFTP servers
running over NFS [50] and PVFS [29].

We note that the current status of our GridFTP/MosaStore prototype implementation
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, since the durable component has no role in serving data to clients
(recall that data is served only by the front-end volatile nodes).

5.4.1. Transfer Throughput

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 18 show that the GridFTP/NFS throughput does not vary with
the stripe width. The reason is the single NFS share which forms a bottleneck at both ends.
The middle bars present the performance of the GridFTP/PVFS server. Each DTP process accesses PVFS through a PVFS client module that offers a traditional file system interface; hence, as discussed previously, introducing an extra layer that hides data locality and adds extra overhead 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 enables exploiting data locality and offers full parallelism: 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.

Figure 18: Average achieved throughput while transferring 1GB file, and varying the stripe width.

5.4.2. Scalability

To assess the performance of our design under high load, we setup a GridFTP server supported by 10 storage nodes, and accessed by 40 clients running on 10 different machines.
Clients are started at 5s intervals and each client reads 100 files of 100MB each. Figure 19 shows the throughput aggregated over all clients.

Figure 19: GridFTP server throughput for 40 clients reading 100 files of 100MB each. The GridFTP server is supported by 10 storage nodes each connected at 1Gbps.

This experiment 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: GridFTP/Mosastore achieves 60% increase in aggregate throughput compared to out-of-the-box (yet still optimized in terms of configuration parameters) GridFTP/PVFS setup.
6. Conclusions

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 we used to evaluate several aspects related to this hybrid architecture, and are summarized as follows:

1.) The effect of having one replica stored on a medium with low access rate (i.e., a durable component). Adding a back-end durable component brings a number of gains: First, the durable component offers strong durability guarantees for the whole system. This is because the system becomes resilient to fatal concurrent failures (i.e., failures that hit all the replicas of a set of objects), the biggest threat to the durability of replicated storage systems.

Second, adding a durable component dramatically reduces replica maintenance costs, while offering the same durability guarantees. In particular, the presence of a durable component halves maintenance traffic generated as well as the disk space used, further it reduces the required peak replication bandwidth by almost one order of magnitude.

Finally, these advantages come at the cost of the additional complexity of an asymmetric data replication scheme and slightly lower availability.
2.) *The impact of resource constraints on availability and replica maintenance costs.* We evaluated the effect of three system parameters: replication bandwidth at the durable as well as volatile nodes, and the replication level.

On the one hand, increasing the *durable component’s bandwidth* has limited contribution in improving availability; on the other hand, increasing the *volatile nodes’ bandwidth* improves average availability at the cost of allowing higher peak replication bandwidth. Finally, increasing the *replication level* improves both the average and minimum availability at the cost of higher maintenance traffic as well as peak replication bandwidth.

3.) *The effect of replica placement scheme on data availability.* We evaluated 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 prototype use-case application, we demonstrate that our approach can reduce the cost of data-access infrastructures while maintaining their high-performance characteristics. Our evaluation of a prototype GridFTP server that integrates volatile storage proves the high-throughput potential of the system and its ability to scale under heavy workload.
References


