Comparative Analysis of Distributed Storage Systems: Architectural Design, Performance, and Cost Trade-offs in Modern Cloud Environments

Pramath Parashar

Data Science Specialist, BHP Minerals Service Company
srivatsa.pramath@gmail.com
ORCID: 0009-0000-4902-7958

Abstract

With the exponential growth of data and increasing need for remote access, distributed storage systems (DSSes) have emerged as a foundational technology for efficient data management and scalability. Traditional storage models are challenged by the requirements of Internet-based systems, where reliability, scalability, and cost-efficiency are essential. This study explores the transition from proprietary hardware-based systems to cloud- based storage solutions and the pressures driving organizations toward Storage as a Service (STaaS) models. Emphasizing the cost-saving and scalability benefits of using commodity hardware, this paper examines three open- source distributed storage systems—Tahoe Least-Authority Filesystem (Tahoe-LAFS), Ceph, and Lustre—each representing a unique architectural approach to data distribution and management. Our research evaluates these systems based on key metrics, including scalability, availability, durability, performance, dynamic operation, and cost-effectiveness. Through analysis and implementation of these DSSes in a networked environment, we aim to understand the design trade-offs and performance characteristics inherent in each system, providing insights into the best practices for organizations transitioning to modern, scalable, and resilient storage architectures.

Index Terms: Distributed Storage Systems (DSS), Storage as a Service (STaaS), Cloud Storage, Scalability, Availability, Performance, Tahoe-LAFS, Ceph, Lustre, Commodity Hardware, Data Durability, Open-Source Storage, Networked Storage Architecture.

I. Problem Definition

Distributed Storage Systems (DSSes) integrate networking and storage to extend the functionality of traditional storage solutions. With the rise of Internet-based storage, DSSes face challenges such as network delays, unreliable communication, and security threats. To address these, modern DSSes leverage commodity hardware for scalability and cost efficiency, moving away from proprietary solutions that struggle to scale with growing data demands.

Storage management has shifted from centralized decision-making to a more application-centric ap- proach, driven by the adoption of cloud and virtualization strategies. However, full migrations remain impractical for larger organizations, leading to hybrid tiered storage models. Meanwhile, storage capacity continues to grow at over 20% annually, while utilization rates remain low, typically around 40%. This highlights the need for improved storage management practices.

These trends create both opportunities and challenges, as new solutions emerge to enhance efficiency, security, and accessibility. Our project examines and implements three open-source DSSes to explore trade-offs in designing scalable and cost-effective storage solutions.

II. Introduction

DSSes have evolved from basic remote storage to offering advanced services such as data distribution,

organization, and security. The rapid expansion of network infrastructure has outpaced processor and storage advancements, driving innovations in DSSes.

Tahoe - Write Data										
RANDOM										
dd-write										
input	random - 1MB	random - 10MB	random - 100MB	random - 1GB	random - 1GB	random - 2GB	random - 2GB	random - 2GB	random - 3GB	random - 3GB
block size (bytes)	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024
blocks	1024	10240	102400	1048576	1048576	2097152	2097152	2097152	3145728	3145728
total size (bytes)	1048576	10485760	104857600	1073741824	1073741824	2147483648	2147483648	2147483648	3221225472	3221225472
total size (KiB)	1024	10240	102400	1048576	1048576	2097152	2097152	2097152	3145728	3145728
total size (MiB)	1	10	100	1024	1024	2048	2048	2048	3072	3072
total size (GiB)	0.001	0.010	0.098	1	1	2	2	2	3	3
dd total time (self) (s)	0.98	10.07	102.09	977.85	1,374.69	2,126.54	1,987.46	1,963.39	2,984.87	2,973.64
dd throughput (b/s)	1.07E+06	1.04E+06	1.03E+06	1.10E+06	7.81E+05	1.01E+06	1.08E+06	1.09E+06	1.08E+06	1.08E+06
dd throughput (KiB/s)	1.04E+03	1.02E+03	1.00E+03	1.07E+03	7.63E+02	9.86E+02	1.06E+03	1.07E+03	1.05E+03	1.06E+03
dd throughput (MiB/s)	1.02E+00	9.93E-01	9.80E-01	1.05E+00	7.45E-01	9.63E-01	1.03E+00	1.04E+00	1.03E+00	1.03E+00
dd throughput (GiB/s)	9.94E-04	9.70E-04	9.57E-04	1.02E-03	7.27E-04	9.40E-04	1.01E-03	1.02E-03	1.01E-03	1.01E-03
dd start time (date)	5/4/2012 21:29:59	5/4/2012 21:35:45	5/4/2012 21:38:27	4/25/2012 15:59:04	4/25/2012 17:25:01	4/30/2012 15:18:03	5/2/2012 9:38:35	5/4/2012 12:43:17	5/2/2012 10:48:27	5/4/2012 14:16:02
dd start time (seconds)	77399	77745	77907	57544	62701	55083	34715	45797	38907	51362
dd end time (date)	5/4/2012 21:30:01	5/4/2012 21:35:55	5/4/2012 21:40:10	4/25/2012 16:15:23	4/25/2012 17:47:58	4/30/2012 15:53:32	5/2/2012 10:11:44	5/4/2012 13:16:02	5/2/2012 11:38:14	5/4/2012 15:05:39
dd end time (seconds)	77401	77755	78010	58523	64078	57212	36704	47762	41894	54339
Tahoe end time (date)	5/4/2012 21:30:02	5/4/2012 21:36:04	5/4/2012 21:41:22	4/25/2012 16:26:56	4/25/2012 18:01:56	4/30/2012 16:21:29	5/2/2012 10:34:36	5/4/2012 13:39:04	5/2/2012 13:42:16	5/4/2012 17:14:03
Tahoe end time (seconds)	77402	77764	78082	59216	64916	58889	38076	49144	49336	62043
Total Time (seconds)	3	19	175	1672	2215	3806	3361	3347	10429	10681
dd Total Time (seconds)	2	10	103	979	1377	2129	1989	1965	2987	2977
Tahoe Total Time (seconds)	1	9	72	693	838	1677	1372	1382	7442	7704
dd Throughput (MBps)	1.02	0.99	0.98	1.05	0.74	0.96	1.03	1.04	1.03	1.03
Tahoe Throughput (MBps)	1.00	1.11	1.39	1.48	1.22	1.22	1.49	1.48	0.41	0.40
Total Throughput (MBps)	0.33	0.53	0.57	0.61	0.46	0.54	0.61	0.61	0.29	0.29
ZEROS										
dd-write										
input	zeros - 1MB	zeros - 10MB	zeros - 100MB	zeros - 1GB	zeros - 1GB	zeros - 1GB	zeros - 1GB	zeros - 2GB	zeros - 2GB	zeros - 3GB
block size (bytes)	1024	1024	1024	1024	1024	1024	1024	1024	1024	1024
blocks	1024	10240	102400	1048576	1048576	1048576	1048576	2097152	2097152	3145728
total size (bytes)	1048576	10485760	104857600	1073741824	1073741824	1073741824	1073741824	2147483648	2147483648	3221225472
total size (KiB)	1024	10240	102400	1048576	1048576	1048576	1048576	2097152	2097152	3145728
total size (MiB)	1	10	100	1024	1024	1024	1024	2048	2048	3072
total size (GiB)	0.001	0.010	0.098	1	1	1	1	2	2	3
dd total time (self) (s)	0.90	8.36	87.28	1,145.64	1,155.84	876.29	1,074.01	2,630.58	1,787.42	2,676.60
dd throughput (b/s)	1.16E+06	1.25E+06	1.20E+06	9.37E+05	9.29E+05	1.23E+06	1.00E+06	8.16E+05	1.20E+06	1.20E+06
dd throughput (KiB/s)	1.14E+03	1.22E+03	1.17E+03	9.15E+02	9.07E+02	1.20E+03	9.76E+02	7.97E+02	1.17E+03	1.18E+03
dd throughput (MiB/s)	1.11E+00	1.20E+00	1.15E+00	8.94E-01	8.86E-01	1.17E+00	9.53E-01	7.79E-01	1.15E+00	1.15E+00
dd throughput (GiB/s)	1.08E-03	1.17E-03	1.12E-03	8.73E-04	8.65E-04	1.14E-03	9.31E-04	7.60E-04	1.12E-03	1.12E-03
dd start time (date)	5/4/2012 21:51:20	5/4/2012 21:53:15	5/4/2012 21:55:29	4/25/2012 13:00:52	4/25/2012 13:29:24	4/25/2012 15:22:08	5/1/2012 14:21:26	4/30/2012 17:19:30	5/3/2012 15:25:40	5/3/2012 17:25:29
dd start time (seconds)	78680	78795	78929	46852	48564	55328	51686	62370	55540	62729
dd end time (date)	5/4/2012 21:51:21	5/4/2012 21:53:24	5/4/2012 21:56:57	4/25/2012 13:19:58	4/25/2012 13:48:41	4/25/2012 15:36:45	5/1/2012 14:39:27	4/30/2012 18:03:24	5/3/2012 15:55:29	5/3/2012 18:10:08
dd end time (seconds)	78681	78804	79017	47998	49721	56205	52767	65004	57329	65408
Tahoe end time (date)	5/4/2012 21:51:22	5/4/2012 21:53:31	5/4/2012 21:58:11	4/25/2012 13:23:25	4/25/2012 13:54:52	4/25/2012 15:38:59	5/1/2012 16:08:06	4/30/2012 20:06:03	5/3/2012 16:00:00	5/3/2012 19:29:30
Tahoe end time (seconds)	78682	78811	79091	48205	50092	56339	58086	72363	57600	70170
Total Time (seconds)	2	16	162	1353	1528	1011	6400	9993	2060	7441
dd Total Time (seconds)	1	9	88	1146	1157	877	1081	2634	1789	2679
Tahoe Total Time (seconds)	1	7	74	207	371	134	5319	7359	271	4762
dd Throughput (MBps)	1.11	1.20	1.15	0.89	0.89	1.17	0.95	0.78	1.15	1.15
Tahoe Throughput (MBps)	1.00	1.43	1.35	4.95	2.76	7.64	0.19	0.28	7.56	0.65
Total Throughput (MBps)	0.50	0.63	0.62	0.76	0.67	1.01	0.16	0.20	0.99	0.41

Fig. 1: The Tahoe topology [1]

Cloud computing has accelerated the shift from server-linked to distributed storage, offering cost reductions but raising security concerns, especially with Storage-as-a-Service (STaaS). While STaaS benefits small to mid-sized businesses, larger organizations retain hybrid storage models due to contractual obligations.

Emerging trends include thin provisioning, which optimizes storage allocation and power consumption, and automated tiering, which assigns data based on usage patterns. As data storage needs grow with Web 3.0 and semi-structured data, DSSes must address challenges like scalability, fault tolerance, and consistency. This paper explores current DSS solutions, highlighting key trade-offs shaping the future of storage.

III. Related Work

Appropriated capacity research, beginning during the 1980s, has developed from fundamental issues to cutting edge points like distributed frameworks, information lattices, and difficulties in directing, consistency, security, and league. Late examinations analyze replication and eradication codes in enormous scope P2P frameworks, with [3] showing that deletion codes lessen data transmission yet add intricacy.

Frameworks like NFS, Coda, and Ivy support unpredictable distribute/share highlights, while execution driven DSSs like PVFS [4], Shine [5], and GPFS [6] take care of I/O-escalated applications.

Custom frameworks like GFS [7] and OceanStore [8], [9] address explicit requirements involving P2P for adaptability. Freeloader [10] enhances data transmission through capacity searching.

We next investigate the structures of Tahoe-LAFS [11], Ceph [12], and Gloss [5], featuring decentralized, adaptable methodologies.

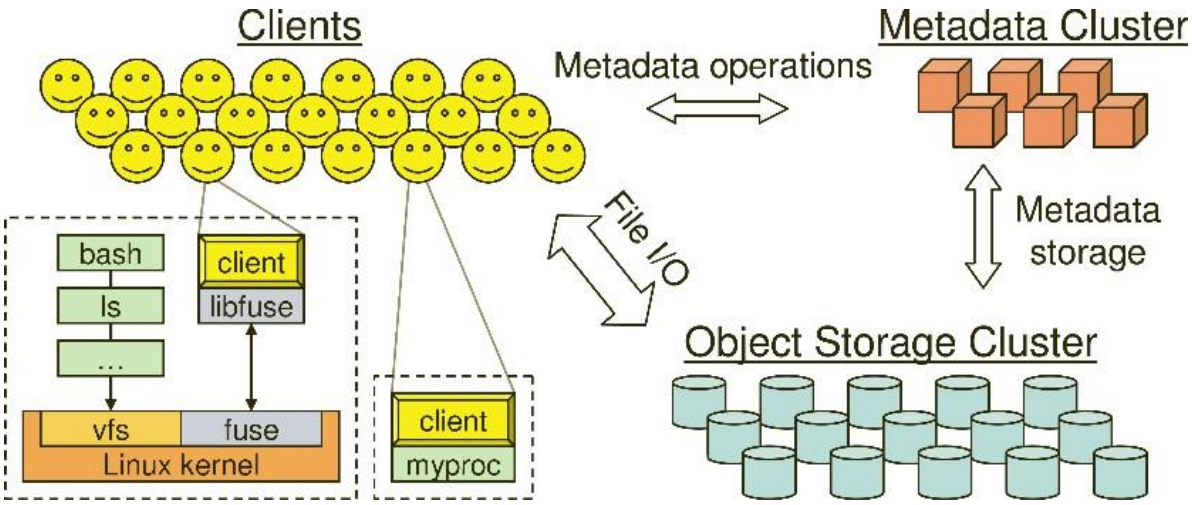

Fig. 2: The Ceph Topology [2]

IV. Background

We evaluated three distributed storage systems—Tahoe, Ceph, and Lustre—based on scalability, avail- ability, durability, performance, and cost-efficiency.

A. Tahoe

Tahoe is an open-source, peer-to-peer file system using encryption and erasure coding for data integrity. It employs a capability-based access model with immutable (read-only) and mutable (read-write) files. Directories enable hierarchical access control, and revocation is done by copying data to a new location.

1) Architecture: Tahoe consists of: 1. A Distributed Hash Table (DHT) for encrypted storage. 2. A Virtual Drive for structured data organization. 3. RESTful and local filesystem interfaces.

While reliable, Tahoe relies on a central introducer node and lacks data location control, making it ideal for local but not wide-area networks.

B. Ceph

Ceph is a scalable, high-performance DSS eliminating single points of failure. It includes: - Clients with a POSIX-like API. - Object Storage Daemons (OSDs) for data management. - Metadata Servers (MDSs) for namespace consistency.

1) Architecture: Ceph separates metadata and data, using the CRUSH algorithm for replication. Storage is unified under RADOS with automated failure recovery. Though highly scalable, Ceph faces challenges in WAN deployments due to network variability.

C. Lustre

Lustre, used in high-performance computing, separates metadata and data, utilizing: - Metadata Servers (MDSs) for namespace management. - Object Storage Servers (OSSs) storing data across Object Storage Targets (OSTs).

1) Architecture: Lustre enables parallel file access by striping data across OSTs. It supports high-performance tuning but suffers from metadata bottlenecks and lacks built-in replication, limiting its use in large-scale distributed environments.

TABLE I: System Comparison

Attribute	Tahoe	Ceph	Lustre
License	GNU GPL	GNU LGPL	GNU GPL
Data Primitive	object (file)	object (file)	object (file)
Data Placement	random	placement groups, pseudo-random mapping	based on round robin and free space heuristics
Metadata Handling	flat	multiple metadata servers	max of two metadata servers
Storage Tiers	none	through CRUSH rules	pools of object storage tar-gets
Failure Handling	assuming unreliable nodes	assuming unreliable nodes	assuming reliable nodes
Replication	server side	server side	server side
WAN Deployment	numerous	no known deployment	TeraGrid (scientific data)
Client Interfacing	The API exposes standard GET, PUT, POST, and DELETE methods, supports JSON and HTML output	native client file system, FUSE	native client file system, FUSE, clients may export NFS, CIFS
Node Types	client/server, an introducer, a key generator	client, metadata, object	client, metadata, object

TABLE II: Tahoe Testbed Machines

Name	CPU	Memory	HD	OS
xpc1	Intel P4 3.00 GHz	2×512MB DDR400	DiamondMax10 160GB SATA	Ubuntu 11.10 x64
xpc2	Intel Celeron D 2.93 GHz	1×1GB DDR400	DiamondMax10 160GB SATA	Ubuntu 11.10 x64
xpc3	AMD Sempron 3000+	2×512MB DDR400	DiamondMax10 160GB SATA	Ubuntu 11.10 x64

V. Test Setup

Circulated capacity frameworks might work on committed machines or client workstations, confronting difficulties from changing circumstances like organization execution, machine network, and information access recurrence. These frameworks' presentation and asset use rely upon arrangement boundaries and dynamic circumstances.

Execution is commonly estimated by information recovery/capacity time, with network assets frequently being the bottleneck. The arrangement of the information recovery system, especially the level of simul- taneousness, influences execution. A low simultaneousness degree can prompt high variety in recovery times, while a high simultaneousness degree can cause network bottlenecks. Changing simultaneousness in light of these elements enhances both execution and asset utilization.

Given the powerful idea of these frameworks, programmed design changes are liked over static settings. The accompanying subsections depict the underlying test arrangement and setup for the frameworks being assessed, barring ensuing boundary changes.

A. Tahoe Test Environment

The Tahoe test environment consist of three micro-ATX PCs. Their characteristics are available in Table II. Each machine is running the latest desktop version of Ubuntu 11.10 as of the time of this writing (Early April, 2012). Each machine has an identical 160GB SATA 3.0 Gb/s hard disk partitioned in the following manner using an MBR partition table:

- **sdX1** - 1 GB - primary - Swap partition
- **sdX2** - 60 GB - primary - ext4 OS partition
- **sdX3** - 100 GB - Extended partition table for LBA partitions
- **sdX5** - 30 GB - logical - ext4 - Tahoe Data partition
- **sdX6** - 30 GB - logical - TBD - Unused
- **sdX7** - 30 GB - logical - ext4 - Unused

The figure shows our testbed setup. All machines are connected to each other via a 10/100 IPv4 Ethernet network connected to a Cisco Linksys WRT110 Switch. The network is local and shared only by the three

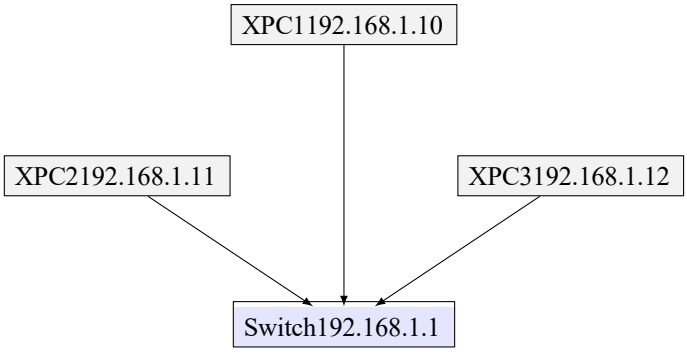

Fig. 3: Tahoe Testbed Setup

test machines and the switch. The switch also acts as a router, gateway and firewall, regulating traffic to and from the Internet. The machines use static IP addresses:

- **xpc1** - 192.168.1.10
- **xpc2** - 192.168.1.11
- **xpc3** - 192.168.1.12
- **Switch** - 192.168.1.1

When running tests, we attempted to insure that each machine was performing no extra work other than the work required to run the benchmark and the backing file system under test. We also attempted to separate the behavior and performance of the various file systems under test to avoid cross-effects between the systems. Thus, only one file system was tested at a time, and all other file system were isolated to their own partition and shutdown while these tests were occurring.

XPC1 is serves as both a Tahoe Introducer, and a Tahoe data node. XPC2 and XPC3 serve only as headless data nodes. XPC1 also serves hosts our primary interface to the Tahoe system: the Fuse SSHFS filesystem. This system provides a POSIX VFS compatible mount point into teh Tahoe system, allowing us to run tests as though it were a standard Unix filesystem.

B. Ceph and Lustre Test Environment

The Lustre/Ceph tests were conducted in a separate test environment from Tahoe due to the need for very specific resources.The objective in the evaluation was to evaluate a few straightforward use cases in a wide area setup with a limited number of nodes. We realize that a limited test environment as used here is not capable of supporting a broad number of scenarios and limits the conclusions that can be drawn particularly with respect to performance behavior across a more industrial strength implementation as well as related issues like fault tolerance; such a test would require a very large setup and considerably more resources to execute than we could secure for this project. Notwithstanding that fact, the test environment is briefly described below.

The experimental evaluation of our system was conducted in a cluster of 14 identical PCs with dual core CPUs running at 1:2 GHz, 1 GB RAM, 1 HDD spinning at 7200 rpm and 1 Gbps Ethernet cards. All PCs were directly connected to a Gigabit Ethernet switch (HP ProCurve 1810g). The bandwidth measure- ment benchmarks conducted in the evaluation environment using netperf indicate a maximum data rate between two PCs (full-duplex) of about 800 Mbps. For the Ceph system tests, we used the Ubuntu 10:04 distribution. For Lustre tests we used Ubuntu 8:10 with the patched kernel provided at the Lustre website.

C. Tahoe - Establishment and Configuration

We set up the Tahoe LAFS on three Ubuntu 11.10 PCs with a parcel mounted at/Tahoe.

Lustre Modification

Design Problem and Challenges

This module is a modification to Lustre to address issues that exist with the file system's metadata server (MDS). Currently the file system provides only a standby architecture for the MDS. The current approach exposes the system to a single point of failure as it only uses one shared disk for its two MDSs. The intent of the module is to implement a process that replicates the MDS on several nodes, thus providing a higher degree of MDS availability. Note that baseline Lustre provides no replication capability, which is a severe weakness in its architectural design.

Some direction to how this architectural change might be effected can be found by borrowing from the symmetric active/active architecture for the MDS of PVFS which provides some design guidance for a similar type issue. In that instance an internal replication of the MDS was implemented with the use of Transis as a group communication facility. In Lustre one can get similar results by isolating the MDS from the other system components and then replicating the MDS. The MDS replication can be done either externally or internally. A detailed discussion of the pros and cons of each is available at the Lustre wiki.

For a number of reasons we chose to replicate externally. This approach constructs the group communication system like a wrapper around the MDS. The group communication system is placed into the Client-MDS communication path as an intermediate communication process. This process intercepts the network calls to and from the MDS and distributes the TCP packages to the group communication facility. The obvious benefit of this approach is that there is no need to touch the MDS code. Conversely, this approach will more than likely result in higher latency time due to the added overhead of inter-process communication. Additionally, the exact communication protocol and format between the MDS and the client must be known. This is an issue since the Lustre code manuals and wiki are uncharacteristically silent on this issue.

Finally, this modification requires a slight reconfiguration of Lustre from the baseline configuration approach. In this modified configuration, Lustre and the network components are configured in a single XML file. In this way the file system
assumes that every node in the file system uses the same XML file for the configuration and startup. However, there appears to be no significant issue in utilizing different XML files for separate nodes. A detailed discussion of the reconfiguration process is not provided here due to space limitations.

Another critical issue in this redesign that needs to be addressed is the single instance execution problem. Since MDS members operate in synchrony each MDS has to generate the same output. The group communication software needs to be

Fig. 4: The Lustre topology [13]

1) Download the most recent Tahoe LAFS variant (e.g., 1.9.1) from [1] and remove it on every PC.
2) Guarantee Python 2.7.2 is introduced (expected for Tahoe LAFS 2.4.4 or higher) [14].
3) Assemble Tahoe LAFS utilizing the order:

```
python setup.py assemble
```

4) Pick a PC (XPC1) to be the speaker because of better execution. Set up subdirectories for speaker and client on XPC1.
5) Make the speaker hub utilizing:

```
tahoe make speaker - - hub directory=/tahoe/speaker
```

Alter the `tahoe.cfg` to design the moniker, web port, and offers.
6) Start the speaker with:

```
tahoe start - - hub directory=/tahoe/speaker/
```

7) On the client hubs (XPC2 and XPC3), make hubs with:

```
tahoe make hub - - hub directory=/tahoe/
```

On XPC1, make the client hub:

```
tahoe make hub - - hub directory=/tahoe/client/
```

8) Duplicate the speaker Roll into the clients' `tahoe.cfg` document and arrange the moniker and offers.
9) Start the client hubs with: `tahoe start/tahoe/` On XPC1, use:

```
tahoe start/tahoe/client/
```

We confirmed the arrangement by getting to the web interface at `127.0.0.1:3457` on XPC1 and transferring/downloading documents from the client interface at `127.0.0.1:3456`. We affirmed that all hubs were associated and working accurately.
For testing comfort, we composed contents to remotely begin/stop the hubs properly aligned, are given in Supplement.

VI. Ceph - Establishment and Configuration

Ceph prescribes `btrfs` for capacity because of its versatility and exchange support. The principal arrangement record `/and so on/ceph/ceph.conf` is shared across hubs. It upholds both `btrfs` and `ext4`, with `ext4` requiring extraordinary mounting choices. SSH keys should be set up for remote access. Ceph utilizes a piece driver or wire to mount and deal with full OSDs by rebalancing or expanding edges. Backup replay mode recuperates MDS disappointments by storing metadata.

VII. Lustre - Establishment and Configuration

Radiance requires Linux bit 2.6.16+ and a fixed portion for MDS, MGS, or OSS. `LNET` organizing and `MGS` setup are fundamental. OSTs are made with an order and should indicate the `MGS` IP address. After establishment, Gloss is mounted and gotten to by clients, with discretionary high accessibility arrangements like Assault and failover.

VIII. Evaluation

The quick advancement of capacity innovation is thwarted by the absence of comparing estimation devices. The shortfall of normalized assessment strategies and deficient measurements from capacity sellers makes it hard for clients to choose the right items, prompting terrible showing and squandered speculations. Slow capacity execution brings about longer question times and unfortunate client encounters, while wasteful utilization of quick stockpiling prompts pointless spending.
Execution testing for circulated capacity frameworks is mind boggling, impacted by variables, for example, block size, document size, line profundity, read/compose proportion, circle speed, reserve size, Attack level, and capacity convention. Thusly, a solitary benchmark is deficient, and various benchmarks are required for precise and solid testing.
The testing stage plans to assess the tradeoffs in framework plan and what these decisions mean for execution. The Linux test stage surveys document framework I/O execution (total transmission capacity, simultaneous access, and metadata throughput) and circle exhibit I/O execution (IOPS, DTR), with adaptable boundaries, for example, record size, block size, line profundity, and read/compose proportions. The testing system incorporates choosing benchmarks, arranging boundaries, picking the objective circulated document framework and test hubs, running the tests, showing results, and creating reports.
Test results and designs are put away in a data set for simple correlation.
To oversee and screen tests, the stage should uphold dynamic expansion/evacuation of test hubs, guaranteeing versatility and effective errand the board.

A. Tahoe Tests

We assessed the Tahoe framework's throughput and idleness utilizing the Unix `dd` utility, with timing from GNU `time`, `date`, and Tahoe's inherent instrumentation. Tests included record sizes from 1 MiB to 3 GiB and both irregular (`/dev/urandom`) and zero-filled (`/dev/zero`) information. We estimated read and compose throughput, and compose dormancy.
For read and compose throughput, we utilized:

```
date; dd if=<Tahoe File> of=/dev/invalid bs=1024; date; date; dd
if=/dev/urandom of=
```

```
<Tahoe File> bs=1024 count=<size>; date;
```

Compose idleness was determined from the time between the fruition of the `dd` compose and record accessibility in Tahoe.

B. Ceph and Gloss Tests

We evaluated the I/O execution of Ceph and Brilliance utilizing the IOzone device with consecutive and arbitrary access designs. Frameworks had 13 capacity hubs, one metadata server, and one client, with a 1 MB block size. We likewise tried versatility with synchronized clients utilizing the `xdd` instrument, and assessed replication execution. Ceph and Tahoe support replication, with Ceph's bigger stripe size affecting execution.

IX. RESULTS

Here we discuss the relevant results from each of our three tests systems.

A. Tahoe Results

The full set of Tahoe read and write results are available. We present an illustrative subset of the results here. Table III shows write throughput speeds for files composed of random data. The average end-to-end write throughput using random data is 0.52 MiB/s.

TABLE III: Tahoe `random` Write Throughput by File Size

Data	Size	Base	Overhead	End-to-End
random	1 MiB	1.02 MiB/s	1.00 MiB/s	0.51 MiB/s
random	10 MiB	0.99 MiB/s	1.11 MiB/s	0.53 MiB/s
random	100 MiB	0.98 MiB/s	1.39 MiB/s	0.57 MiB/s
random	1 GiB	1.05 MiB/s	1.48 MiB/s	0.61 MiB/s
random	2 GiB	1.03 MiB/s	1.49 MiB/s	0.61 MiB/s
random	3 GiB	1.03 MiB/s	0.41 MiB/s	0.29 MiB/s

Table IV shows write throughput speeds for files composed of zeros data. The average end-to-end write throughput using zeros data is 0.69 MiB/s.

TABLE IV: Tahoe `zeros` Write Throughput by File Size

Data	Size	Base	Overhead	End-to-End
zeros	1 MiB	1.11 MiB/s	1.00 MiB/s	0.50 MiB/s
zeros	10 MiB	1.20 MiB/s	1.43 MiB/s	0.63 MiB/s
zeros	100 MiB	1.15 MiB/s	1.35 MiB/s	0.62 MiB/s
zeros	1 GiB	1.17 MiB/s	7.64 MiB/s	1.01 MiB/s
zeros	2 GiB	1.15 MiB/s	7.56 MiB/s	0.99 MiB/s
zeros	3 GiB	1.15 MiB/s	0.65 MiB/s	0.41 MiB/s

As we can see from the write data, it appears that Tahoe's compression capabilities are kicking on for the larger 'zeros' files. Random files are not generally compressible, so no such effect is seen there. Files exceeding 2 GiB also seem to suffer a significant performance hit. This may be due to limits imposed by the amount of RAM available on our test machines. The large files may be forcing the SSHFS system to use a lot of swap-space since each machine was only equipped with 1GB of RAM, leading to a performance crash above a certain file size.

Table V shows read throughput speeds for files composed of random data. The average read throughput using random data is 1.32 MiB/s.

TABLE V: Tahoe `random` Read Throughput by File Size

Data	Size	Base
random	1 MiB	1.54 MiB/s
random	10 MiB	1.32 MiB/s
random	100 MiB	1.11 MiB/s

TABLE VI: Tahoe `zeros` Read Throughput by File Size

Data	Size	Base
zeros	1 MiB	1.61 MiB/s
zeros	10 MiB	1.53 MiB/s
zeros	100 MiB	1.13 MiB/s

Table VI shows read throughput speeds for files composed of zeros data. The average read throughput using zeros data is 1.42 MiB/s.

Tahoe read throughput tends to be about twice that of write throughput. This is likely due to a combination of the fact that Tahoe must propagate write changes across the network, as well as the fact that the SSHFS interface must cache the file before uploading it. No read data was collected for files larger than 100 MiB. The system became unstable when we attempted to read files of this size. Again, this may be a limitation of the system RAM size and the SSHFS interface.

X. Ceph and Shine Results Summary

Figures show performance across nonconcurrent I/O workloads for Lustre256, Lustre0, and Ceph. As file sizes grow, throughput stabilizes due to reduced caching. Lustre0 outperforms Ceph and Lustre256 in sequential writes but is limited by network bandwidth (95 MB/s). Ceph excels in larger file stripes but incurs higher file operation costs.

For concurrent writes, Lustre scales up to 300 MB/s, while Ceph peaks at 180 MB/s. Sequential reads and writes remain stable at larger request sizes, reaching 330 MB/s for reads. Ceph saturates uplinks during asynchronous writes, while Tahoe performs better in direct writes but degrades with additional replicas.

XI. General Conclusions

Traditional DFS solutions (e.g., Lustre, HDFS) rely on a single metadata server, limiting scalability. Newer systems like Ceph distribute metadata but neglect network traffic optimization. Future DFS should minimize inter-server communication, leverage on-demand data retrieval, and reassess data redundancy strategies. A topology-aware striping mechanism could enhance performance in multi-application envi- ronments.

A. System-Specific Conclusions

Tahoe: Secure against untrusted storage nodes through encryption but suffers from slow writes, making it suitable for archival storage.

Ceph: Scalable with dynamic metadata distribution and replication but hindered by immaturity, limited documentation, and WAN performance concerns.

Lustre: Highly performant in HPC environments but bottlenecked by a single metadata server and lacks native replication.

B. Comparison

Table VII provides a side-by-side comparison of the three systems. Tahoe is best for untrusted envi- ronments, Lustre for HPC workloads, and Ceph for a modern POSIX-compliant storage system. Tahoe's encryption overhead impacts performance, while Ceph and Lustre achieve higher throughput.

TABLE VII: Comparison of Distributed Storage Systems

Feature	Tahoe	Lustre	Ceph
Throughput	Low	High	High
Latency	High	Low	Low
Fault Tolerance	User-defined	Failover	Replication
Security	High	Moderate	High
Installation	Dist.	Dist.	Dist.
Management	Dist.	Dist.	Dist.
Compatibility	Multi-OS	Linux	Linux
Data Placement	Random	Heuristics	Groups
Storage Tiers	None	OSTs	CRUSH
Maintained	Yes	Yes	Yes
Development	Yes	Yes	Yes

XII. Future Work

Scaling tests across multiple geographically distributed sites would provide better insights into real-world WAN performance. Future experiments should incorporate diverse hardware setups (SSD, SAS, RAID) and network technologies (Ethernet, InfiniBand). Enhancements could include a native FUSE interface for Tahoe, improved metadata redundancy in Lustre, and streamlined deployment tools for Ceph.

References

[1] T.-L. Team", ""tahoe-lafs website"," http://www.tahoe-lafs.org, 2012.
[2] J. B. Laton", ""ceph: The distributed file system creature from the object lagoon"," 2010.
[3] H. Weatherspoon and J. D. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," *Revised Papers from the 1st Intl. Workshop on Peer-to-Peer Systems*, vol. 2429, pp. 328–337, Dec 2002.
[4] P. H. Carns, W. B. L. III, R. B. Ross, and R. Thakur, "Pvfs: A parallel file system for linux clusters," *Extreme Linux Workshop*, 2000.
[5] P. Braam, "Lustre white paper," 2005.
[6] F. R. H. Schmuck, "Gpfs: A shared-disk file system for large computing clusters," *Proceedings of the FAST'02 Conference on File and Storage Technologies*, Jan 2002.
[7] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 29–43, 2003.
[8] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: An architecture for global-scale persistent storage," *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Nov 2000.
[9] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: the oceanstore prototype," *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*, Mar 2003.
[10] S. Vazhkudai, X. Ma, V. Freeh, J. Strickland, N. Tammineedi, and S. Scott, "Freeloader:scavenging desktop storage resources for scientific data," *Proceedings of Supercomputing 2005 (SC'05)*, 2005.
[11] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pp. 21–26, 2008.
[12] S. A. Weil, "Ceph: Reliable, scalable, and high-performance distributed storage," *Dissertation*, 2007. [Online]. Available: http://ceph.newdream.net/weil-thesis.pdf
[13] "http://lxhzju.blog.163.com/", ""lustre file system","
http://lxhzju.blog.163.com/blog/static/450082006102164962 89/, 2006.
[14] P. D. Team", ""python website"," http://www.python.org, 2012.
[15] A. Fox, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, 2009.
[16] C. Kang, "A distributed storage schema for cloud computing based raster gis systems," pp. 1–19, May 2011.
[17] B. Trushkowsky, P. Bodık, A. Fox, M. Franklin, M. Jordan, and D. Patterson, "The scads director: Scaling a

distributed storage system under stringent performance requirements," *USENIX Conf on File and Storage Technologies*, pp. 163–176, 2011.
[18] C. Maltzahn, E. Molina-Estolano, A. Khurana, A. Nelson, S. Brandt, and S. Weil, "Ceph as a scalable alternative to the hadoop distributed file system," *USENIX; login*, vol. 35, no. 4, pp. 38–49, 2010. [Online]. Available: http:
//www.usenix.org/publications/login/2010-08/openpdfs/maltzahn.pdf
[19] J. Feng, Y. Chen, and P. Liu, "Bridging the missing link of cloud data storage security in aws," pp. 1–2, Oct 2009.
[20] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," *Proceedings of the 7th symposium on Operating systems design and implementation*, pp. 307–320, 2006.
[21] Q. He, Z. Li, and X. Zhang, "Study on cloud storage system based on distributed storage systems," pp. 1332–1335, Dec 2010.
[22] K. Liu, J. Zhou, L. Qin, and N. Lv, "A novel computing-enhanced cloud storage model supporting combined service aware," pp. 275–280, Jul 2010.
[23] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," pp. 44–51, Aug 2009.
[24] C. Wang, Q. Wang, K. Ren, and W. Lou, "Towards secure and dependable storage services in cloud computing," *Services Computing, IEEE Transactions on*, no. 99, pp. 1–1, 2011.
[25] Y. Han, "Cloud computing: Case studies and total cost of ownership," *Information Technology and Libraries*, Jan 2011. [Online].
Available: http://ejournals.bc.edu/ojs/index.php/ital/article/view/1871
[26] U. Divakarla. . . , "An overview of cloud computing in distributed systems," *American Institute of Physics . . .*, Jan 2010. [Online].
Available: http://adsabs.harvard.edu/abs/2010AIPC.1324..184D
[27] A. Marinos and G. Briscoe, "Community cloud computing," *Cloud Computing*, pp. 472–484, 2009.
[28] L. Wang, J. Tao, M. Kunze, D. Rattu, and A. Castellanos, "The cumulus project: Build a scientific cloud for a data center," *Cloud Computing and its Applications*, 2008.
[29] J. Hansen and E. Jul, "Lithium: virtual machine storage for the cloud," *Proceedings of the 1st ACM symposium on Cloud computing*,
pp. 15–26, 2010.
[30] S. Smaldone, C. Tonde, L. Iftode, V. K. Ananthanarayanan, and A. Elgammal, "The cyber-physical bike: A step towards safer green transportation." [Online]. Available: http://www.research.rutgers.edu/~smaldone/pubs/cyberbike-hotmobile11.pdf
[31] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 1–26, Jun 2008.
[32] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*. [Online]. Available: http://dl.acm.org/citation.cfm?id=359563
[33] C. Hsu[1], C. Wang, and S. Shieh, "Reliability and security of large scale data storage in cloud computing."
[34] H. Chihoub, G. Antoniu, and M. S. Perez, "Towards a scalable, fault-tolerant, self-adaptive storage for the clouds," 2011.
[35] R. Hegarty, M. Merabti, Q. Shi, and B. Askwith, "Forensic analysis of distributed data in a service oriented computing platform," *PG Net The 10th Annual Postgraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting, Liverpool John Moores University*, 2009.
[36] T. B. Winans and J. S. Brown, "Cloud computing a collection of working papers," pp. 1–33, Jul 2009.
[37] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems." [Online]. Available: http://dl.acm.org/citation.cfm?id=214456
[38] C. Wang, Q. Wang, K. Ren, and W. Lou, "Ensuring data storage security in cloud computing," *Quality of Service, 2009. IWQoS. 17th International Workshop on*, pp. 1–9, 2009.
[39] H. Yoon, M. Ravichandran, A. Gavrilovska, and K. Schwan, "Distributed cloud storage services with flecs containers."
[40] A. Muthitacharoen, R. Morris, T. Gil, and B. Chen, "Ivy: A read/write peer-to-peer file system," *5th OSDI*, 2002.