# High Available Data Placement in Large-scale Distributed Storage System

Myat Pwint Phyu
*University of Computer Studies, Yangon*
*myatpwint.ucsy@gmail.com*

## Abstract

*Today's storage systems have a great challenge for the long-term storage of massive amounts of unstructured data. Availability and reliability is the properties of the most storage system. In this paper, the data distribution in large scale distributed storage system have been presented. To provide high data availability, replicas will be placed with Chord-based replication method with extra weight metric.*

## 1. Introduction

As storage system grows larger and more complex, the traditional file systems cannot satisfy the large work load. Many file systems have emerged focusing on specialized requirements such as data sharing, remote file access, distributed file access, parallel files access, high performance computing, archiving etc.. Moreover, storing and managing the growth of unstructured data is one of the great challenges.

Data availability is one of the most important factors in a distributed environment. In the first part of the figure 1, (100/6) ~17% of the files will distribute in the file system onto each server. Any single file will be stored on a single storage server. It cannot stand any failure. In the second part, (100/3) ~33% of the files will distribute over mirror in the file system onto each mirror pair. Any single file will be stored on two storage servers. Complete replication or parity scheme are used to achieve this. Replication is the simplest redundancy scheme. It forces extremely high bandwidth and storage overhead.

With an erasure-coded redundancy scheme, each object is divided into m fragments and recoded into n fragments, where n>m. So, the effective redundancy factor is k=n/m. With erasure codes, the original object can be reconstructed from any m fragments.

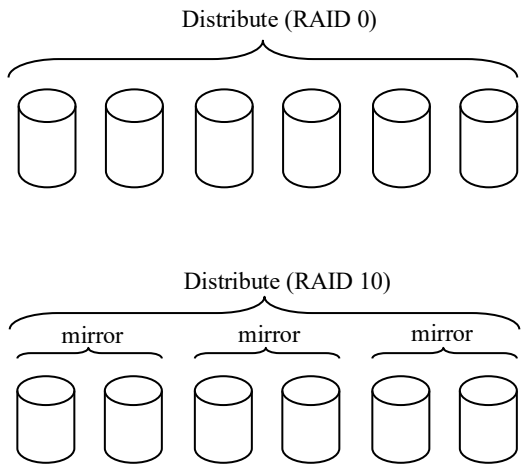


**Figure 1. Distribution vs. replication**

In this paper, we will discuss how replicas are placed in large scale distributed storage system. We will adjust access frequency and storage capacity using Chord mechanism to achieve efficient scalability, availability and cost reduction.

The rest of this paper is organized as follows. Section 2 shows the related works of the system. Section 3 explains the large scale storage system and Section 4 introduces the proposed system. Then we conclude the paper with future work in Section 5.

## 2. Related work

In this section, we will present some of the recent popular large scale file systems and the strategies for the data availability.

GPFS (General Parallel File System) [6] is a high performance shared-disk clustered file system developed by IBM. Files are divided into small blocks (less than 1MB) and blocks are distributed across the cluster. It can support RAID replicated or file system node replicated using a round-robin fashion.

Lustre [10] is a massively parallel distributed file system owned by Oracle. It is OSD based and open source storage system. It can support data rebalance but no replication. Bad metadata performance can only be achieved for a large number of small files. It is difficult to remove and add OSTs elastically.

GlusterFS [11] is a scale out file system for Network Attached Storage (NAS). It is a clustered file system for storing unstructured data. It has no bottlenecks by eliminating metadata servers. It uses an index to look up files, employing elastic hash algorithm to find a unique identifier for each file. It can provide scalability by linearly scales to hundreds of petabytes. It can get high availability with data mirroring and real time self healing. Gluster only provides redundancy at the server level, not at the individual disk level.

Google File System (GFS) [1] and Ceph [7] also use redundant data on different storage nodes to recover from drive failures. Ceph uses a hash-based distribution scheme, and its object servers propagate replicas to each other.

In the Panasas file system [9], the clustered design of the storage system and the use of client-driven RAID provide scalable performance to many concurrent file system clients through parallel access to file data that is striped across OSD storage nodes.

RUSH [2, 3] and CRUSH [5] are two algorithms for online placement and reorganization of replicated data. They are probabilistically optimal in distributing data evenly and minimizing data movement when new storage is added to the system. But they depend on the existence of a high-quality random function, which is difficult to generate.

A new approach scaling RAID to fulfill the three requirements: uniform data distribution, minimal data migration, fast data addressing. FastScale [8] uses a new and elastic addressing function which moves only enough data blocks from old disks to fill an appropriate fraction of new disks for the first two requirements. There is no data migration between the old disks. FastScale accesses multiple physically successive blocks via a single I/O and records data migration lazily to minimize the number of metadata writes without compromising data consistency. FastScale gives a solution only for RAID 0 and it may also be possible for RAID 10 and RAID 01.

## 3. Large scale shared storage system

The storage system can be classified into three natures. Block storage devices are assumed to have limited intelligence and will read/write whatever is requested. Metadata server has to manage each individual block – overhead quickly mounts up for large number of blocks. For file-based nature, data is striped as files across multiple file servers. It is performed well under many workloads. When the system and the workload scale up, bottleneck may be occurred. In object-based storage, files are broken into smaller chunks called objects. It is identified by unique numbers. Storage system can scale in performance and capacity. Object-based storage is emerging now as another alternative to storing long term unstructured data. It is attractive because of its massive scalability and shared tenancy features, especially in comparison to ordinary file- or block-based storage.

There are two approaches to share data storage. Storage Area Network (SAN) communicates with applications using lower-level protocols, which offer fast performance, but little in the way of intelligent manageability. It identifies data by block number. Network Attached Storage (NAS) identifies information by file type. It utilizes standard protocols such as Network File System (NFS) so data can be stored and accessed more intelligently. However, there may be performance latency trade-off for gaining

that intelligence. It can offer improved manageability on a small scale and suffer performance and management as the volume of file data grows. This can lead inability to scale.

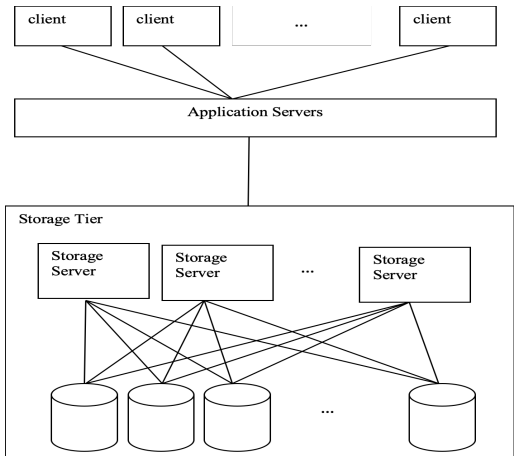


**Figure 2. Architecture of the large scale shared storage system**

By scale up technology, increasing the input/output size and storage capacity of the NAS device as data needs grow. But this has several disadvantages. One is that there may be significant cost for replacing a device with only an incremental increase in capacity. Another is that a single point of failure may still affect. A scale-out design is only limited by the software that combines the many inexpensive servers into one large virtual server. The architecture of large scale shared storage system is shown in figure 2.

# 4. The proposed system

For a distributed storage system focusing data availability, how to implement a redundancy strategy, where to replicate the data and when and how to repair a lost replica must be considered. Generally, data availability increases along with the increase of replica number. But, adding more replicas will not improve the data availability when the replica number reaches a certain point. To maintain minimum replica number ensuring the availability, we will take into account some factors such as access frequency and the node's storage capacity and adjust them efficiently. Then we will use the Chord DHT mechanism to place the replicas.

## 4.1. Chord-based data placement

The use of distributed hash tables has been a natural choice to achieve the scalability goal. Chord DHT protocol [4] provides support for just one operation: given a key, it maps the key onto a node. Node keys are arranged in a circle called Chord ring. The circle cannot have more than $2^m$ nodes. Consistent hash function assigns each node and key an m-bit identifier. SHA-1 is used as a based hash function. As shown in figure 3, key $k$ is assigned to the first node, successor node of the key $k$, whose identifier is equal to or follows $k$ in the identifier space. It is denoted as $successor(k)$. To speed up the lookups, Chord maintains additional routing information which is called finger table. The $i^{th}$ entry in the table at node $n$ contains the identity of the first node s that succeeds $n$ by at least $2^{n-1}$ on the identifier circle.

The weight-based data distribution mechanism is illustrated in figure 3.

```
To find the successor node of an identifier id,
n.find_successor(id)
   while (id ∉ (n, n.successor))
     // to find the closet finger proceeding id
     for i = 1 to m
       if (finger[i].node.weight > max_w)
         max_w = finger[i].node.weight
       if (finger[i].node ∈ (n,id))
         n = finger[i].node
   return n
```

**Figure 3. Data distribution algorithm**

In the proposed method, nodes are organized as the form of ring structure. Data objects are distributed using the Chord mechanism. The conventional Chord mechanism is enhanced by a metric called node's weight. This is determined

by counting the access frequency and adjusting the node's storage capacity. It can also be an aid to determine the addition and deletion of replica data. Each node in the Chord ring must maintain a finger table with extra metric. The metric is not constant and it needs updating at the stabilization routine of the Chord.

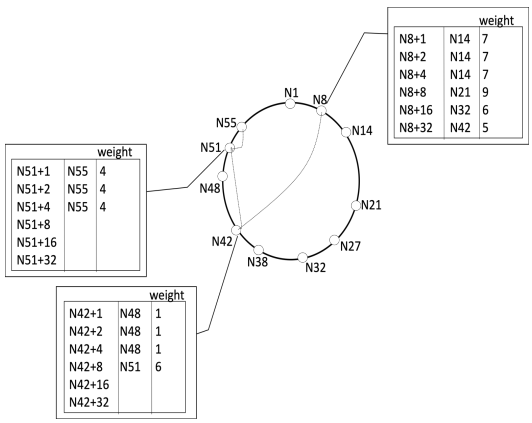


**Figure 4. Chord-based data distribution mechanism**

Figure 4 shows the data distribution mechanism using Chord. When a request comes, Node N8 is randomly chosen for the document of hashed key "52". It looks up in its finger table and forwards the request to the node having the greatest identifier that is less than or equal to "52". It also takes the greatest weight throughout the process. If the process finds the successor node (node N55), it must decide the target node according to the largest weight among the close nodes (node N21).

## 4.2. Replica building and rebuilding

Based on each node's weight, Chord adapts efficiently as nodes join and leave the system and only an $O(1/N)$ fraction of the keys are moved to a different location. When a node n join the network, certain keys previously assigned to n's successor become assigned to n and when node n leaves, all of its assigned keys are just reassigned to n's successor.

## 4.3. Theoretical advantages

To the best of our knowledge, the proposed Chord structure-based data distribution is the efficient one. There are some properties that are offered by the proposed system.

1. The chord mechanism takes the placement of files and optimizes the data distribution as it maintains least requirements for the distribution of data. Since each node holds the finger tables at the power of two intervals around the circle, each node can pass a request at least halfway along the remaining distance which is O (log N).
2. By consistent hashing approach, the nodes can maintain load balancing well because all nodes receive roughly the same number of keys.
3. Because of the weight-based finger table of Chord mechanism, the data can be put over more appropriate node.

# 5. Conclusions and Future Work

In this paper, we present the efficient way to assist the storage management of distributed storage system. We discuss the chord DHT approach for the data placement in large scale distributed storage system. We show that some theoretical advantages. We will evaluate our system with various kinds of workloads later. The system is still in progress and we will present the system with rich experimental results at future.

**References**

[1] S. Ghemawat, H. Gobioff, S. Leung. The Google File System, In Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003, pp 20-43.

[2] R. J. Honicky and E. L.Miller. A fast algorithm for online placement and reorganization of replicated data. In Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), Nice, France, April 2003.

[3] R. J. Honicky and E. L. Miller. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In Proceedings of the

18th International Parallel and Distributed Processing Symposium (IPDPS'04), IEEE. 2004.
[4] L. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer to Peer Lookup Service for Internet Applications. In Proceedings of ACM SIGCOMM 2001, San Diego, August 2001, pp. 160-177.
[5] S. A.Weil, S. A. Brandt, E. L.Miller, and C.Maltzahn. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In Proceedings of the International Conference on Super Computing (SC'06). Tampa Bay, FL. 2006.
[6] F. Schmuck and R. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In Proc. of the First Conference on File and Storage Technologies, 2002.
[7] S. A. Weil, S. A. Brandt, E. Miller, D. Long, C. Maltzahn, Ceph: A Scalable, High-Performance Distributed File System, In Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06), November 2006.
[8] W. Zheng, and G. Zhang, FastScale: Accelerate RAID Scaling by Minimizing Data Migration, In Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST'2011), February 2011.
[9] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, B. Zhou. "Scalable Performance of the Panasas Parallel File System. In Proceedings of the USENIX Conference on File and Storage Technologies (FAST'08), 2008.
[10] Cluster File Systems Inc., Lustre: A scalable high-performance file system, www.lustre.org/documentation.html.
[11] Gluster File System, www.gluster.org.