A Data Rebalancing Mechanism for Gluster File System

Kyar Nyo Aye, Thandar Thein
University of Computer Studies, Yangon, Myanmar
kyarnyoaye@gmail.com, thandartheinn@gmail.com

Abstract

Data rebalancing is one of the most interesting research areas in distributed file systems. In Gluster file system, data among the storage servers are rebalanced after adding a new storage server to the Gluster storage pool or removing the storage server from the Gluster storage pool. The main issue in Gluster file system is inefficient data rebalancing; a large number of file migrations, a large amount of files migration time and inefficient storage utilization. Therefore, a data rebalancing mechanism for Gluster file system is proposed to achieve efficient storage utilization, to reduce the number of file migrations and to save files migration time. There are two main contributions in this paper: using consistent hashing algorithm with virtual nodes from Amazon dynamo to reduce the number of file migrations and to save files migration time and migration of virtual nodes between storage servers to provide efficient storage utilization. The proposed data rebalancing mechanism and current data rebalancing mechanism are simulated with Java and the proposed mechanism provides 82% (fullness percent), 20% of the number of file migrations, 20% of the files migration time, and 73% of the number of required storage servers of the current mechanism of Gluster file system.

Keywords: data rebalancing, file migration, Gluster file system, storage utilization

1. Introduction

Data rebalancing [2] is the ability to auto-balance the system after adding or removing servers. Tools must be provided to recover lost data, to store them on other servers or to move them from a hot device to a newly added one. Communications between machines allow the system to detect servers' failures and servers overload. To correct these faults, servers can be added or removed. When a server is removed from the system, the latter must be able to recover the lost data, and to store them on other servers. When a server is added to the system, tools for moving data from a hot server to the newly added server must be provided. Users do not have to be aware of this mechanism. Usually, distributed file systems use a scheduled list in which they put data to be moved or recopied. Periodically, an algorithm iterates over this list and performs the desired action.

In Gluster file system (GlusterFS) [4], storage server is added to increase storage capacity and storage server is removed to reduce storage capacity. And then the data among storage servers are needed to rebalance to ensure uniform data distribution. Data rebalancing is one of the major problems in GlusterFS. There are some issues in current data rebalancing mechanism in GlusterFS. The first issue is a large number of file migrations in rebalancing process. The second issue is a large amount of files migration time. The third issue is inefficient storage utilization – a large number of required storage servers and under utilization of these storage servers. A data rebalancing mechanism for GlusterFS is required to address these problem issues.

The aim of this paper is to propose a data rebalancing mechanism for GlusterFS to achieve efficient storage utilization, to reduce the number of file migrations and to save files migration time. The rest of the paper is organized as follows: In section 2, we present related work and explain background theory such as GlusterFS, data rebalancing in GlusterFS and dynamo in section 3. In section 4, we introduce our proposed data rebalancing mechanism for GlusterFS. Then conclusion is described in section 5.

2. Related Work

We survey some of existing data rebalancing mechanisms in distributed file systems: Hadoop distributed file system (HDFS) [5, 6, 11], general parallel file system (GPFS) [9, 10], and Google file system (GFS) [3, 8]. A HDFS cluster can easily become imbalanced, when a new data node joins the cluster. Because it does not hold much data, any map task assigned to the machine most likely does not access local data, thus increasing the use of network bandwidth. On the other hand, when some data nodes become full, new data blocks are placed on only non-full data nodes, thus reducing their read parallelism. It is important to redistributed data blocks when imbalance occurs. A HDFS cluster is balanced if there is no under-utilized or above-utilized data node in the HDFS cluster. A data node's utilization is defined as the percentage of its used space. The rebalancing server makes rebalancing decision iteration by iteration. At each iteration the major goal is to make every over/under-utilized data node less imbalanced rather than reducing the number of imbalanced data nodes. In this way rebalancing is able to maximize the use of the network bandwidth.

Adding a disk to a file system can be a regular occurrence if the file systems are rapidly growing in size, and free disk space is needed. GPFS gives the option to add disks to the file system online, while the cluster is active, and the new empty space can be used as soon as the command completes. When a new disk is added using the mmadddisk command with the -r balance flag, all existing files in the file system will use the new free space. A new disk can be added without rebalancing, and do a complete rebalancing later, with the mmrestripefs command.

A Google file system cluster consists of a single master and multiple chunkservers and is accessed by multiple clients. Files are divided into fixed-size chunks. For reliability, each chunk is replicated on multiple chunkservers. Chunk replicas are created for three reasons: chunk creation, re-replication, and rebalancing. When the master creates a chunk, it chooses where to place the initially empty replicas. The master re-replicates a chunk as soon as the number of available replicas falls below a user-specified goal. Finally, the master rebalances replicas periodically: it examines the current replica distribution and moves replicas for better disks pace and load balancing. Also through this process, the master gradually fills up a new chunkserver rather than instantly swamps it with new chunks and the heavy write traffic that comes with them. In addition, the master must also choose which existing replica to remove. In general, it prefers to remove those on chunkservers with below-average free space so as to equalize disk space usage.

3. Background Theory

This section provides an overview of GlusterFS, data rebalancing in GlusterFS, and Dynamo [1, 7].

3.1 GlusterFS

The GlusterFS is an open source distributed file system that can scale out in building-block fashion to store multiple petabytes of data. The clustered file system pools storage servers over TCP/IP or InfiniBand Remote Direct Memory Access (RDMA), aggregating disk and memory and facilitating the centralized management of data through a unified global namespace. The software works with low-cost commodity computers. GlusterFS supports standard clients running standard applications over any standard IP network.

Figure 1 illustrates how users can access application data and files in a Global namespace using a variety of standard protocols. GlusterFS gives users the ability to deploy scale-out, virtualized storage – scaling from terabytes to petabytes in a centrally managed and commoditized pool of storage.

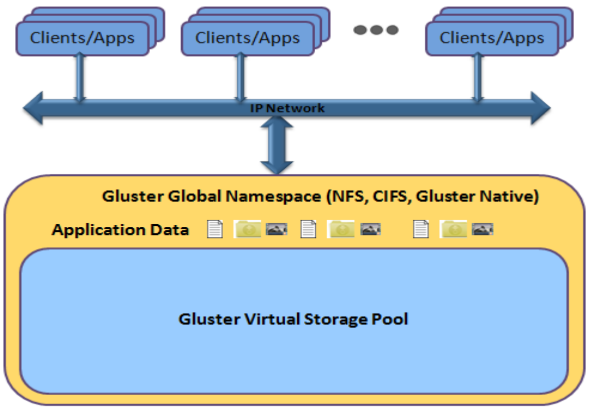
Clients/Apps
Clients/Apps
Clients/Apps
IP Network
Gluster Global Namespace (NFS, CIFS, Gluster Native)
Application Data
Gluster Virtual Storage Pool

Figure 1. GlusterFS – One Common Mount Point

Scale-out storage systems based on GlusterFS are suitable for unstructured data such as documents, images, audio and video files, and log files.

3.2 Data Rebalancing in GlusterFS

Data rebalancing, this is one of the key challenges – and therefore one of the most interesting research areas. The first thing to know about GlusterFS rebalancing is that it's not automatic. If a new brick is added, even new files will not be put on it until the "fix-layout" part of rebalance is done, and old files will not be put on it until the "migrate-data" part is done.

- Fix-layout just walks the directory tree recalculating and modifying the trusted.glusterfs.dht.xattrs to reflect the new list of bricks. It does this in a pretty simple way, assigning exactly one range of length MAXINT/nbricks to each brick in turn starting at zero.
- Migrate-data is much more costly. For each file, it calculates where the file should be according to the new layout. Then, if the file is not already there, it moves the file by copying and renaming over the original. There's some tricky code to make sure this is transparent and atomic and so forth, but that's the algorithm.

There are enhancement opportunities in both of these areas. While the migrate-data issue is at the level of mechanics and implementation, the fix-layout issue is at more of a conceptual level. When a new brick is added, approximately 1/new_brick_count hash values should be reallocated. Because layout calculations are naïve, much more than that will be reallocated – exacerbating the migrate-data problem because reallocated hash values correspond to moved files.

3.3 Dynamo – A Distributed Storage System

Unlike a relational database, Dynamo is a distributed storage system. Like a relational database it stores information to be retrieved, but it does not break the data into tables. Instead all objects are stored and looked up via a key. Figure 2 shows a distributed storage system.

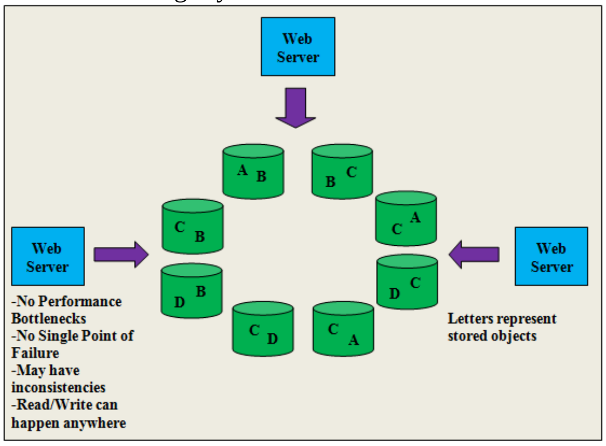
Web Server
Web Server
Web Server
A B
B C
C A
D C
C A
C D
D B
C B
-No Performance Bottlenecks
-No Single Point of Failure
-May have inconsistencies
-Read/Write can happen anywhere
Letters represent stored objects

Figure 2. Distributed Storage System

3.3.1 Working Process of Dynamo

Like Amazon S3, Dynamo offers a simple put and get interface. Each put requires the key, context and the object. Here is the high level description of Dynamo and a put request:

- Physical nodes are thought of as identical and organized into a ring.
- Virtual nodes are created by the system and mapped onto physical nodes, so that hardware can be swapped for maintenance and failure.
- The partitioning algorithm is one of the most complicated pieces of the system; it specifies which nodes will store a given object.
- The partitioning mechanism automatically scales as nodes enter and leave the system.
- Any node in the system can issue a put or get request for any key.

So Dynamo is quite complex, but is also conceptually simple. It is inspired by the way things work in nature – based on self-organization and emergence. Each node is identical to other nodes, the nodes can come in and out of existence, and the data is automatically balanced around the ring.

3.3.2 Partitioning in Dynamo

One of the key design requirements for Dynamo is that it must scale incrementally. This requires a mechanism to dynamically partition the data over the set of nodes (i.e., storage hosts) in the system. Dynamo’s partitioning scheme relies on consistent hashing to distribute the load across multiple storage hosts. In consistent hashing, the output range of a hash function is treated as a fixed circular space or “ring” (i.e. the largest hash value wraps around to the smallest hash value). Each node in the system is assigned a random value within this space which represents its “position” on the ring. Each data item identified by a key is assigned to a node by hashing the data item’s key to yield its position on the ring, and then walking the ring clockwise to find the first node with a position larger than the item’s position. Thus, each node becomes responsible for the region in the ring between it and its predecessor node on the ring. The principle advantage of consistent hashing is that departure or arrival of a node only affects its immediate neighbors and other nodes remain unaffected.

The basic consistent hashing algorithm presents some challenges. First, the random position assignment of each node on the ring leads to non-uniform data and load distribution. Second, the basic algorithm is oblivious to the heterogeneity in the performance of nodes. To address these issues, Dynamo uses a variant of consistent hashing (instead of mapping a node to a single point in the circle, each node gets assigned to multiple points in the ring. To this end, Dynamo uses the concept of “virtual nodes”. A virtual node looks like a single node in the system, but each node can be responsible for more than one virtual node. Effectively, when a new node is added to the system, multiple positions (henceforth, “tokens”) are assigned in the ring.

Using virtual nodes has the following advantages:

- If a node becomes unavailable (due to failures or routine maintenance), the load handled by this node is evenly dispersed across the remaining available nodes.
- When a node becomes available again, or a new node is added to the system, the newly available node accepts a roughly equivalent amount of load from each of the other available nodes.
- The number of virtual nodes that a node is responsible can be decided based on its capacity, accounting for heterogeneity in the physical infrastructure.

3.3.3 Ensuring Uniform Load Distribution

Dynamo uses consistent hashing to partition its key space across its replicas and to ensure uniform load distribution. A uniform key distribution can help us achieve uniform load distribution assuming the access distribution of keys is not highly skewed. In particular, Dynamo’s design assumes that even where there is a significant skew in the access distribution there are enough keys in the popular end of the distribution so that the load of handling popular keys can be spread across the nodes uniformly through partitioning. There are three partitioning strategies in Dynamo and the proposed data rebalancing mechanism uses Dynamo’s partitioning strategy 1.

Strategy 1: T random tokens per node and partition by token value: In this scheme, each node is assigned T tokens (chosen uniformly at random from the hash space). The tokens of all nodes are ordered according to their values in the hash space. Every two consecutive tokens define a range. The last token and the first token form a range that "wraps" around from the highest value to the lowest value in the hash space. Since the tokens are chosen randomly, the ranges vary in size. As nodes join and leave the system, the token set changes and consequently the ranges change.

Strategy 2: T random tokens per node and equal sized partitions

Strategy 3: Q/S tokens per node, equal-sized partitions

4. Proposed Data Rebalancing Mechanism for Gluster File System

This section describes the proposed data rebalancing mechanism for GlusterFS. In this section, the proposed data rebalancing mechanism is explained not only for homogeneous brick sizes but also for heterogeneous brick sizes. When the proposed data rebalancing mechanism is compared with current data rebalancing mechanism, we found that the former mechanism has smaller file migration percentages than the latter mechanism.

4.1 Proposed Data Rebalancing Mechanism for Homogeneous Brick Sizes

This section explains how to handle homogeneous brick sizes in current and the proposed data rebalancing mechanism. In this section, after a storage server is added or removed, how to change layouts of the hash rings in two data rebalancing mechanisms is illustrated and file migration percentages are calculated according to these changed layouts.

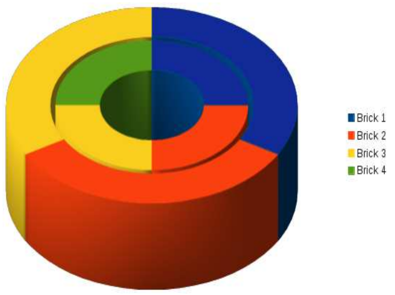

Figure 3. Hash-Range Assignments for Homogeneous Brick Sizes after Adding or Removing a Brick with Current Data Rebalancing Mechanism

Figure 3 shows the hash-range assignments for homogeneous brick sizes after adding or removing a brick with current data rebalancing mechanism. The outer ring represents the state with just three bricks – hash value zero at the top, split into three equal ranges. The inner ring represents the state after adding a fourth brick. Any place where the inner and outer rings are different colors represents a range that has been reassigned from one brick to another – implying a migration of data likewise. Half of the data are being moved when it should be only a quarter- 8% brick1 to brick2, 17% brick2 to brick3, and 25% brick3 to brick4. Similarly, when brick4 is removed from the storage pool which consists of four bricks, half of the data are being moved- 8% brick2 to brick1, 17% brick3 to brick2, and 25% brick4 to brick3.

Instead, the proposed data rebalancing mechanism uses “virtual nodes” concept from Dynamo and assign multiple “virtual node IDs” for brick four, giving it a total of 25% drawn equally from bricks one through three. Quarter of the data are being moved- 8% brick1 to brick4, 8% brick2 to brick4, and 8% brick3 to brick4. Figure 4 shows the hash-range assignments for homogeneous bricks after adding a brick with the proposed data rebalancing mechanism.

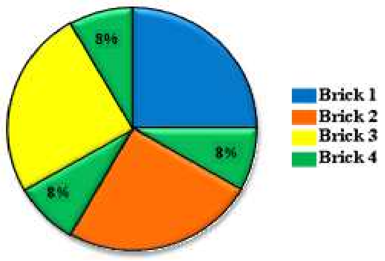

Figure 4. Hash-Range Assignments for Homogeneous Brick Sizes after Adding a Brick with the Proposed Data Rebalancing Mechanism

Similarly, when brick4 is removed, the migration process is reversed and quarter of the data are being moved – 8% brick4 to brick1, 8% brick4 to brick2, 8% brick4 to brick3. Figure 5 shows the hash-range assignments for homogeneous bricks after removing a brick with the proposed data rebalancing mechanism.

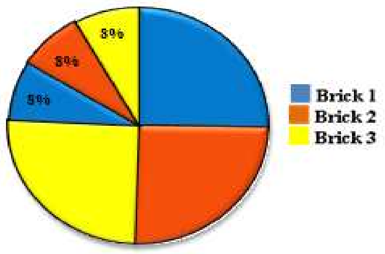

Figure 5. Hash-Range Assignments for Homogeneous Brick Sizes after Removing a Brick with the Proposed Data Rebalancing Mechanism

4.2 Proposed Data Rebalancing Mechanism for Heterogeneous Brick Sizes

The previous cases are only considered for homogeneous brick sizes. To deal with heterogeneous brick sizes, controlled placement with virtual nodes is used in the proposed data rebalancing mechanism. Existing hash-range assignment for three heterogeneous brick sizes is described in Figure 6. In this figure, brick1 (10GB) occupies 17%, brick2 (20GB) occupies 33% and brick3 (30GB) occupies 50% of the total hash space.

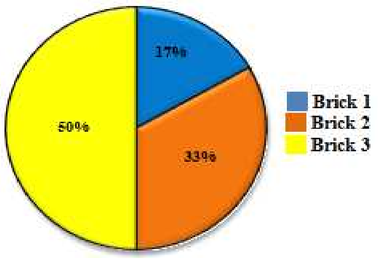

Figure 6. Hash-Range Assignments for Three Heterogeneous Brick Sizes

Figure 7 shows the existing hash-range assignment for four heterogeneous brick sizes. In this figure, brick1 (10GB) occupies 10%, brick2 (20GB) occupies 20%, brick3 (30GB) occupies 30%, and brick4 (40GB) occupies 40% of the total hash space.

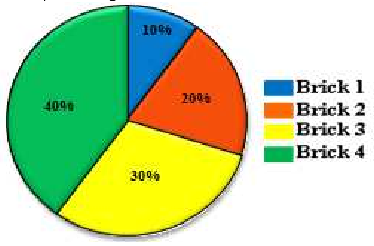

Figure 7. Hash-Range Assignments for Four Heterogeneous Brick Sizes

In current data rebalancing mechanism, when brick4 is added, 67% of the data are being moved – 7% brick1 to brick2, 20% brick2 to brick3, 40% brick3 to brick4. Figure 8 shows the hash-range assignments for heterogeneous bricks after adding a brick with current data rebalancing mechanism.

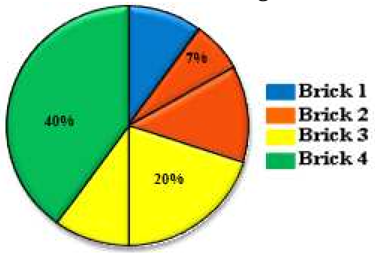

Figure 8. Hash-Range Assignments for Heterogeneous Brick Sizes after Adding a Brick with Current Data Rebalancing Mechanism

Similarly, when brick4 is removed, 67% of the data are being moved- 7% brick2 to brick1, 20% brick3 to brick2, and 40% brick4 to brick3. Figure 9 shows the hash-range assignments for heterogeneous bricks after removing a brick with current data rebalancing mechanism.

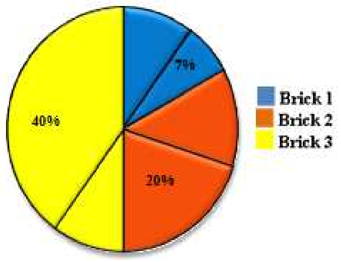

Figure 9. Hash-Range Assignments for Heterogeneous Brick Sizes after Removing a Brick with Current Data Rebalancing Mechanism

In the proposed data rebalancing mechanism, when brick4 is added, 40% of the data are being moved – 7% brick1 to brick4, 13% brick2 to brick4, 20% brick3 to brick4. Figure 10 shows the hash-range assignments for heterogeneous bricks after adding a brick with the proposed data rebalancing mechanism.

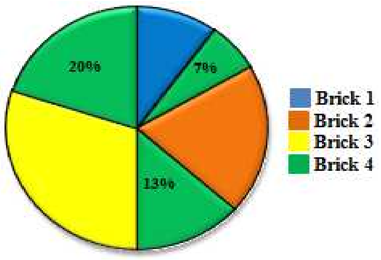

Figure 10. Hash-Range Assignments for Heterogeneous Brick Sizes after Adding a Brick with the Proposed Data Rebalancing Mechanism

Similarly, when brick4 is removed, 40% of the data are being moved- 7% brick4 to brick1, 13% brick4 to brick2, and 20% brick4 to brick3.

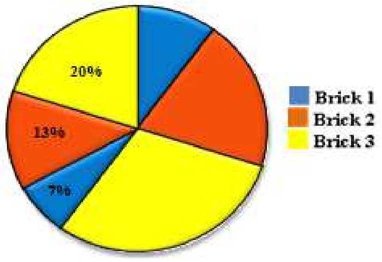

Figure 11. Hash-Range Assignments for Heterogeneous Brick Sizes after Removing a Brick with the Proposed Data Rebalancing Mechanism

Figure 11 shows the hash-range assignments for heterogeneous bricks after removing a brick with the proposed data rebalancing mechanism.

4.3 Adding/Removing Storage Servers to/from Gluster Storage Pool

When a new storage server (server1) is added into the Gluster storage pool, it gets assigned a number of virtual nodes that are randomly scattered on the ring. For every key range that is assigned to server1, there may be a number of nodes that are currently in charge of handling keys that fall within its token range. Due to the allocation of key ranges to server1, some existing nodes no longer have to some of their keys and these nodes transfer those keys to server1. When a node is removed from the system, the reallocation of keys happens in a reverse process.

4.4 Adding/Removing Files to/from Gluster Storage Pool

To add a new file into the Gluster storage pool, file name is hashed by consistent hashing algorithm to get a hash value. Exactly one virtual node has an assigned range including the file's hash value, and so the file is stored on the storage server associated with that virtual node. To remove a file from the Gluster storage pool, file name is hashed by consistent hashing algorithm to get a hash value. Exactly one virtual node has an assigned range including the file's hash value, and so the file is removed from the storage server associated with that virtual node.

4.5 Performance Evaluation

We have evaluated the performance of the two data rebalancing mechanisms with different test cases. There are four GlusterFS operations:

- Adding the storage server to the storage pool
- Removing the storage server from the storage pool
- Adding the file to the storage pool
- Removing the file from the storage pool

GlusterFS is a multi-user clustered file system so multiple users perform different GlusterFS operations on the same GlusterFS server volume concurrently.

4.5.1 Sample Test Cases

Forty test cases are used as sample test cases to evaluate the performance of two data rebalancing mechanisms. In these test cases, multiple users add storage servers and files to the Gluster storage pool, and remove storage servers and files from the storage pool simultaneously. Ten test cases that illustrate the process of concurrently adding storage servers and files are depicted in Table 1.

Table 1 Ten Test Cases for Adding Storage Servers and Files

Test Cases	Operations
Test Case 1	Add 2 Servers Add 1000 files from 1 user
Test Case 2	Add 2 Servers Add 1000 files from 2 users each
Test Case 3	Add 2 Servers Add 1000 files from 3 users each
Test Case 4	Add 2 Servers Add 1000 files from 4 users each
Test Case 5	Add 2 Servers Add 1000 files from 5 users each
Test Case 6	Add 2 Servers Add 1000 files from 6 users each
Test Case 7	Add 2 Servers Add 1000 files from 7 users each
Test Case 8	Add 2 Servers Add 1000 files from 8 users each
Test Case 9	Add 2 Servers Add 1000 files from 9 users each
Test Case 10	Add 2 Servers Add 1000 files from 10 users each

Table 2 shows ten test cases that demonstrate the process of concurrently adding storage servers and files and removing files.

Table 2 Ten Test Cases for Adding Storage Servers and Files and Removing Files

Test Cases	Operations
Test Case 11	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 1 user
Test Case 12	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 2 users each
Test Case 13	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 3 users each
Test Case 14	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 4 users each
Test Case 15	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 5 users each
Test Case 16	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 6 users each
Test Case 17	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 7 users each
Test Case 18	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 8 users each
Test Case 19	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 9 users each
Test Case 20	Add 2 Servers Add 1000 files from 10 users each Remove 1000 files from 10 users each

Ten test cases that simulate the process of simultaneously adding and removing storage servers and files are displayed in Table 3.

Table 3 Ten Test Cases for Adding and Removing Storage Servers and Files

Test Cases	Operations
Test Case 21	Add 2 Servers Add 1000 files from 1 user Remove 2 Servers
Test Case 22	Add 2 Servers Add 1000 files from 1user Remove 1000 files from 1 user
Test Case 23	Add 2 Servers Add 1000 files from 2 users each Remove 2 Servers
Test Case 24	Add 4 Servers Add 1000 files from 3 users each Remove 2 Servers Remove 1000 files from 1 user
Test Case 25	Add 2 Servers Add 1000 files from 4 users each Remove 4 Servers Remove 1000 files from 1 user
Test Case 26	Add 4 Servers Add 1000 files from 3 users each Remove 4 Servers Remove 1000 files from 1 user
Test Case 27	Add 4 Servers Add 1000 files from 4 users each Remove 4 Servers Remove 1000 files from 1 user
Test Case 28	Add 4 Servers Add 1000 files from 4 users each Remove 6 Servers Remove 1000 files from 1 user
Test Case 29	Add 4 Servers Add 1000 files from 4 users each Remove 6 Servers Remove 1000 files from 2 users each
Test Case 30	Add 4 Servers Add 1000 files from 4 users each Remove 6 Servers Remove 1000 files from 3 users each

Table 4 describes ten test cases that illustrate the process of concurrently adding and removing storage servers and files.

Table 4 Next Ten Test Cases for Adding and Removing Storage Servers and Files

Test Cases	Operations
Test Case 31	Add 4 Servers Add 1000 files from 4 users each Remove 6 Servers Remove 1000 files from 4 users each
Test Case 32	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers
Test Case 33	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers Remove 1000 files from 1 user
Test Case 34	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers Remove 1000 files from 2 users each
Test Case 35	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers Remove 1000 files from 3 users each
Test Case 36	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers Remove 1000 files from 4 users each
Test Case 37	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers Remove 1000 files from 5 users each
Test Case 38	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers Remove 1000 files from 6 users each
Test Case 39	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers Remove 1000 files from 7 users each
Test Case 40	Add 2 Servers Add 1000 files from 10 users each Remove 2 Servers Remove 1000 files from 8 users each

4.5.2 Experimental Results for Data Rebalancing

The current data rebalancing mechanism and the proposed data rebalancing mechanism are simulated with Java and performance evaluations are conducted with different test cases in terms of number of file migrations, files migration time, number of required storage servers, storage utilization (fullness percent) and processing time. Figure 12 illustrates the number of file migrations of test case 1 to 10 for the two data rebalancing mechanisms. There is a great deal of difference in the number of file migrations between the two data rebalancing mechanisms. The number of file migrations of test case 10 for the current data rebalancing mechanism is 6 times greater than the number of file migrations for the proposed data rebalancing mechanism. Therefore, Test case 10 has the most significant difference in the number of file migrations between the two mechanisms.

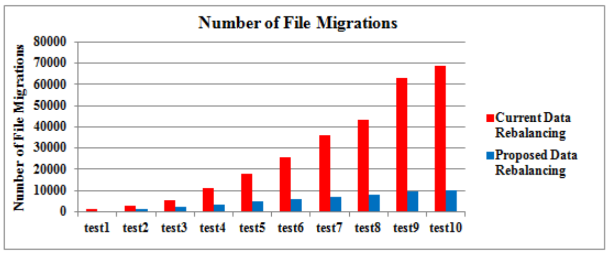

Figure 12 Number of File Migrations of Test Case 1 to 10

Figure 13 shows the number of file migrations of test case 11 to 20 for the two data rebalancing mechanisms. When the more files are added to the storage pool, there are more significant differences in the number of file migrations between the two data rebalancing mechanisms. The number of file migrations of test case 11 for the current data rebalancing mechanism is 6 times greater than the number of file migrations for the proposed data rebalancing mechanism.

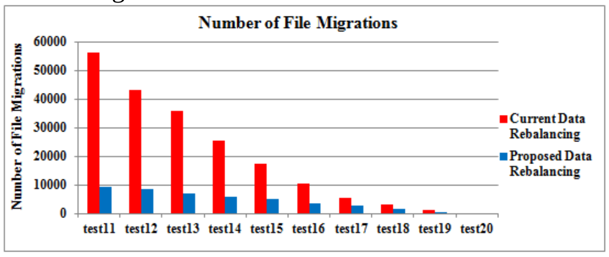

Figure 13 Number of File Migrations of Test Case 11 to 20

Figure 14 displays the number of file migrations of test case 21 to 30 for the two data rebalancing mechanisms. As can be seen, Test case 22 has the least significant difference and test case 28 has the most significant difference in the number of file migrations between the two mechanisms.

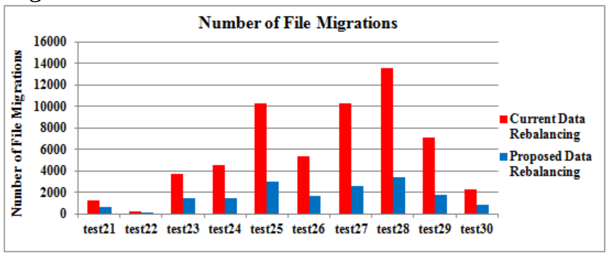

Figure 14 Number of File Migrations of Test Case 21 to 30

Figure 15 describes the number of file migrations of test case 31 to 40 for the two data rebalancing mechanisms. The number of file migrations of test case 32 for the proposed data rebalancing mechanism is 7 times less than the number of file migrations for the current data rebalancing mechanism.

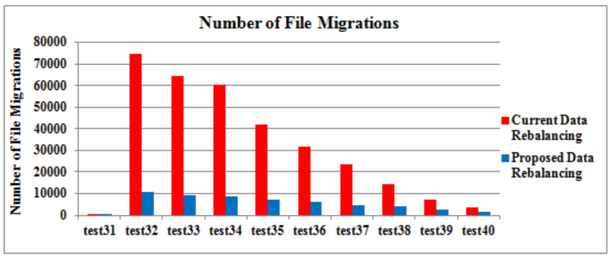

Figure 15 Number of File Migrations of Test Case 31 to 40

The files migration time of test case 1 to 10 for the two data rebalancing mechanisms is illustrated in Figure 16. File migration time depends on the migrated files' sizes. It takes a large amount of time to migrate the many large files. Therefore, the file migration time is directly proportional to the migrated files' sizes. According to Figure 16, Test case 1 has the least significant difference and test case 10 has the most significant difference in the files migration time between the two mechanisms.

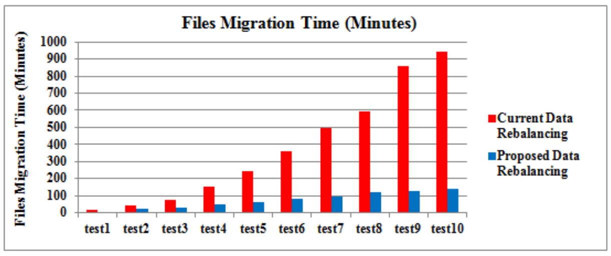

Figure 16 Files Migration Time of Test Case 1 to 10

Figure 17 and Figure 18 show the files migration time of test case 11 to 30 for the two data rebalancing mechanisms.

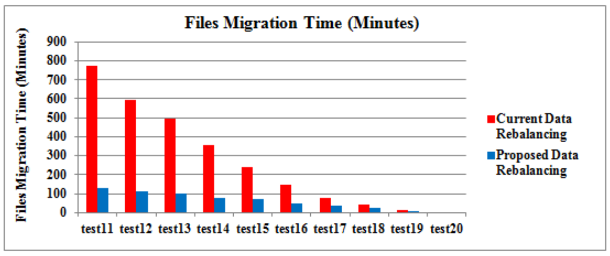

Figure 17 Files Migration Time of Test Case 11 to 20

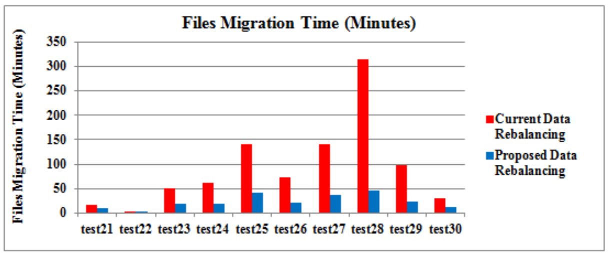

Figure 18 Files Migration Time of Test Case 21 to 30

The files migration time of test case 31 to 40 for the two data rebalancing mechanisms is illustrated in Figure 19. The files migration time of test case 32 for the proposed data rebalancing mechanism is 6.6 times less than the files migration time for the current data rebalancing mechanism.

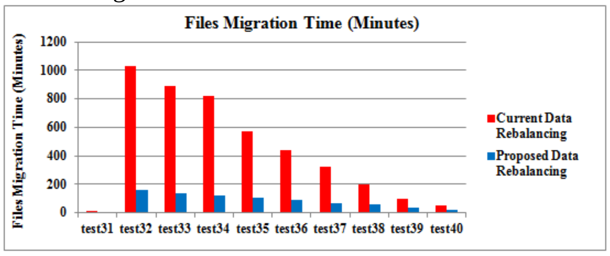

Figure 19 Files Migration Time of Test Case 31 to 40

Figure 20 dictates the number of required storage servers of test case 1 to 10 for the two data rebalancing mechanisms. In test case 1 to 10, 1000 files are added to the storage pool from 1 user to 10 users. The storage pool has 1000 files in test case 1, 2000 files in test case 2, 3000 files in test case 3 and so on. Storage servers are added to the storage pool to store these files. In test case 1, the numbers of required storage servers for the proposed data rebalancing mechanism and current data rebalancing mechanism are 5 and 6. The numbers of required storage servers in test case 10 are 22 and 30 respectively.

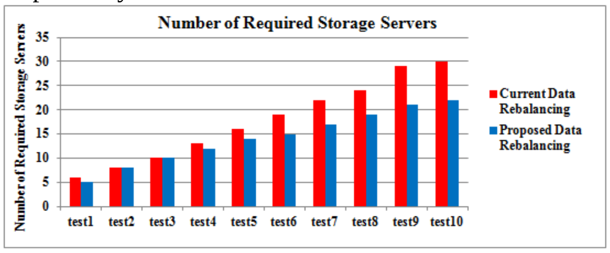

Figure 20 Number of Required Storage Servers of Test Case 1 to 10

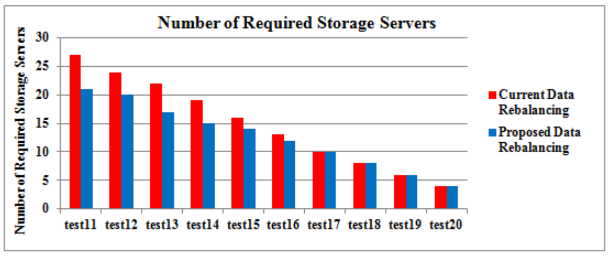

Figure 21 Number of Required Storage Servers of Test Case 11 to 20

Figure 21 describes the number of required storage servers of test case 11 to 20 for the two data rebalancing mechanisms. The number of required storage servers of test case 11 for the proposed data rebalancing mechanism is 1.3 times less than the number of required storage servers for the current data rebalancing mechanism.

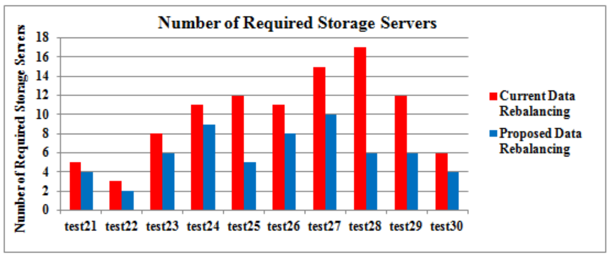

Figure 22 Number of Required Storage Servers of Test Case 21 to 30

The number of required storage servers of test case 21 to 30 for the two data rebalancing mechanisms is shown in Figure 22. The most striking feature is that the number of required storage servers of test case 28 for the proposed data rebalancing mechanism is 3 times less than the number of required storage servers for the current data rebalancing mechanism. Figure 23 displays the number of required storage servers of test case 31 to 40 for the two data rebalancing mechanisms.

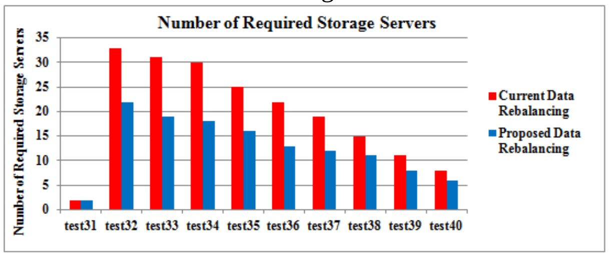

Figure 23 Number of Required Storage Servers of Test Case 31 to 40

Figure 24, 25, 26 and 27 show the storage utilization (fullness percent) of test case 1 to 40 for two data rebalancing mechanisms.

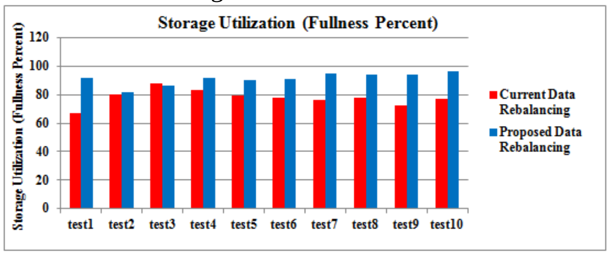

Figure 24 Storage Utilization (Fullness Percent) of Test Case 1 to 10

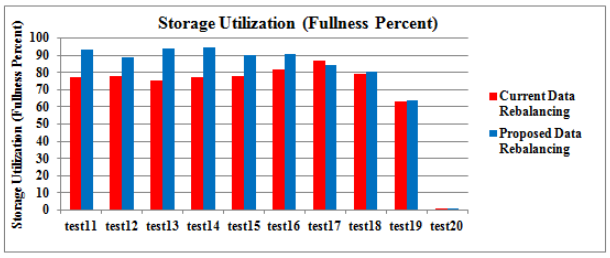

Figure 25 Storage Utilization (Fullness Percent) of Test Case 11 to 20

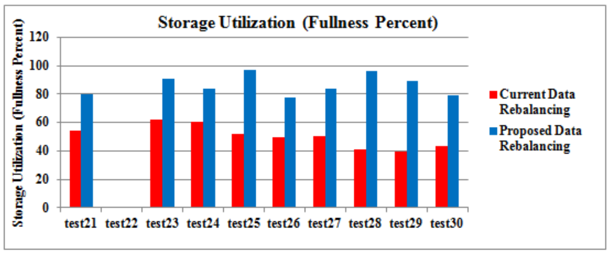

Figure 26 Storage Utilization (Fullness Percent) of Test Case 21 to 30

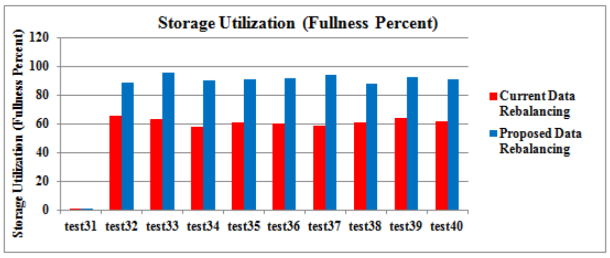

Figure 27 Storage Utilization (Fullness Percent) of Test Case 31 to 40

Figure 28, 29, 30 and 31 display the processing time (milliseconds) of test case 1 to 40 for two data rebalancing mechanisms.

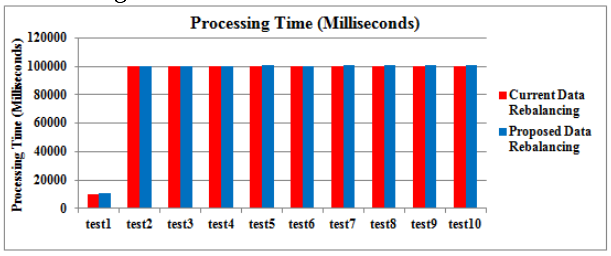

Figure 28 Processing Time of Test Case 1 to 10

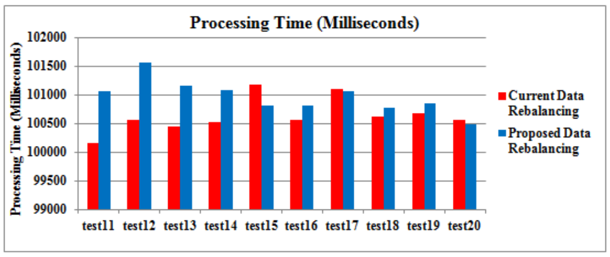

Figure 29 Processing Time of Test Case 11 to 20

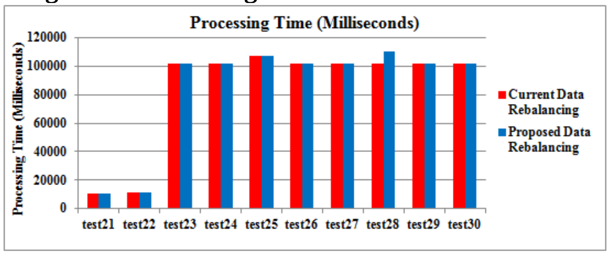

Figure 30 Processing Time of Test Case 21 to 30

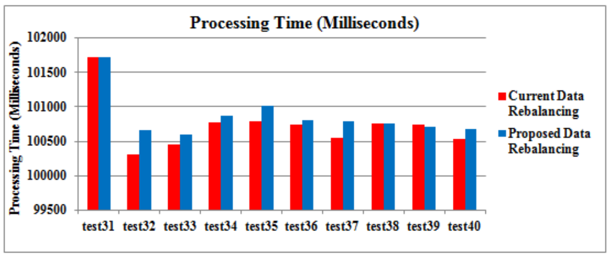

Figure 31 Processing Time of Test Case 31 to 40

5. Conclusion

The main issue in GlusterFS is inefficient data rebalancing; a large number of file migrations, a large amount of files migration time and inefficient storage utilization. Therefore, a data rebalancing mechanism for GlusterFS is proposed to achieve efficient storage utilization, to reduce the number of file migrations and to save files migration time. In this paper, the proposed data rebalancing mechanism is thoroughly described for the combination of two strategies – consistent hashing algorithm with virtual nodes and the migration of virtual nodes to reduce the number of file migrations, to save files migration time, and to achieve efficient storage utilization. The proposed data rebalancing mechanism and current data rebalancing mechanism are simulated with Java and the proposed mechanism provides 82% (fullness percent), 20% of the number of file migrations, 20% of the files migration time, and 73% of the number of required storage servers of the current mechanism of GlusterFS.

References

[1] G. DeCandia, D. Hastorun, M. Jampani, and G. Kakulapati et al., "Dynamo: Amazon's Highly Available Key-value Store", In proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles, Stevenson, WA, USA, October 14-17, 2007, pp. 205-220.
[2] B. Depardon, C. Seguin, G. L. Mahec, "Analysis of Six Distributed File Systems", Available: http://hal.archives-ouvertes.fr/docs/00/78/90/86/PDF/a_survey_of_dfs.pdf, February 15, 2013.
[3] S. Ghemawat, H. Gobioff, S. T. leung, "The Google File System", In Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP' 03), Bolton Landing, NY, USA, October 19-22, 2003, pp. 29-43.
[4] GlusterFS Developers, "Gluster File System 3.3.0 Administration Guide", Available: http://www.gluster.org/wp-content/uploads/2012/05/Gluster_File_System-3.3.0 Administration_Guide-en-US.pdf.
[5] Hadoop, "HDFS Architecture Guide", Available: http://hadoop.apache.org/docs/stable1/hdfs_design.html.
[6] Hadoop, "Rebalance an HDFS Cluster", Available: http://www.webhdd.ru/library/files/RebalanceDesign6.pdf.
[7] A. Iskold, "Amazon Dynamo: The Next Generation of Virtual Distributed Storage", October 30, 2007
[8] J. Passing, "The Google File System and Its Application in MapReduce", Available:

http://int3.de/res/GfsMapReduce/ GfsMapReducePaper.pdf, April 2008.
[9] D. Quintero and M. Barzaghi, "Implementing the IBM General Parallel File System (GPFS) in a Cross-Platform Environment",Available:http://www.redbooks.ibm.com/redbooks/pdfs/sg247844.pdf.
[10] F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters", In Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST' 02), Monterey, California, USA, January 28-30, 2002, pp. 231-244.
[11] K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System", In Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, Incline Village, NV, USA, May 3-7, 2010, pp. 1-10.