A Virtual Shared Metadata Storage for HDFS

Jiang Zhou*†, Yong Chen*, Xiaoyan Gu†, Weiping Wang†, Dan Meng†
*Texas Tech University, Lubbock, TX, USA
†Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{jiang.zhou, yong.chen}@ttu.edu, {guxiaoyan, wangweiping, mengdan}@iie.ac.cn

***Abstract*—Hadoop is a popular open-source framework that allows distributed analysis of large datasets using the MapReduce programming model. A distributed file system HDFS is implemented to provide high-throughput access to datasets. HDFS can achieve high performance metadata service but has two disadvantages. First, when the metadata server stores metadata on persistent devices, it is restricted to read and write operations of local disks. Second, it also lacks effective methods for metadata synchronization and replication, which is critical for metadata availability and reliability. In this research, we introduce a novel Virtual Shared Storage Pool (VSSP) concept and design for storing and sharing metadata in HDFS. The VSSP is a virtual storage device which is built on existing servers and transparent to upper layers. Two strategies, a journal synchronization based on the 2PC protocol and a fine-grained image replication, are introduced in the VSSP according to different metadata access features. The VSSP not only reduces the overhead on metadata modification operations, but also improves the I/O performance for namespace storage. Experimental results show that the VSSP improved the average performance by 40.51% and 23.46% when writing logs compared with the BookKeeper and Hadoop QJM. The average image read and write throughput was nearly 5 times and 2.4 times better than NFS and the original approach. These results confirm that the proposed VSSP solution significantly improves the metadata access performance, scalability, and reliability for HDFS.**

***Index Terms*—Distributed file systems; cluster file systems; HDFS; shared storage; distributed metadata management**

I. Introduction

In recent years we have witnessed a phenomenal increase of data volume and scale in many aspects of computing. A recent survey indicates that the digital universe will grow by a factor of 10 in the next five years and reach 44 zettabytes [1]. Various service requirements and exponential data growth bring many opportunities and challenges. Data-driven scientific discovery has become the fourth paradigm after experiment, theory, and simulation, and the new source of economic growth [2]. Mass data management and analysis have become more important than ever before.

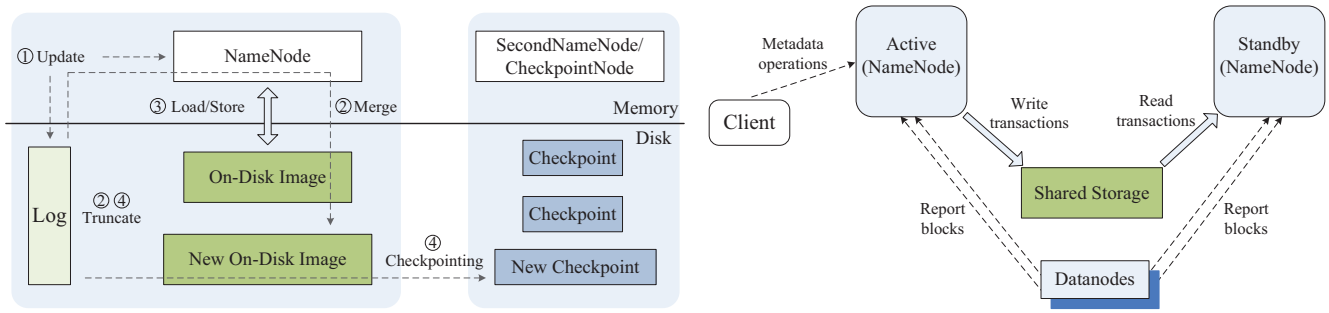
In mass data management and analysis, Apache Hadoop has been the factual standard and widely used for many applications [3]. It implements the MapReduce framework that allows distributed processing of extremely large data sets across clusters of computers using a simple programming model. A distributed file system HDFS is implemented to achieve highly efficient mass data storage solutions [4]. By providing compatible interfaces with Hadoop modules, HDFS provides file system operations for MapReduce jobs and tasks. It adopts the method of decoupling data and metadata processing. It uses a namenode to manage the global namespace and multiple datanodes to store files. Although HDFS provides the capability for mass data storage, there exists two problems in it with the rapid growth of data and cluster scale.

One problem is the metadata scalability and performance for an enormous amount of file and directory accesses. As HDFS uses a single metadata server, it has the performance bottleneck issue with the increase of cluster scale [5], [6]. To improve system scalability, several file systems [7], [8] use multiple metadata servers to manage the global namespace. HDFS relies on the logs for metadata persistence. It is effective but the metadata will be lost partially or entirely for either disk errors, volume failures, or node shutdown. Besides, cross-node transactions [8] may access shared metadata from the global log for keeping server state consistent. It needs an effective metadata distribution strategy to improve the performance of such operations.

The other problem is the metadata reliability for fault tolerance in case of failures. With the growth of cluster scale, the probability of system failures has been significantly increased [9]. For recovery, various reliable mechanisms have been studied in HDFS including a primary/backup strategy [4], hot standby [10], and server state replication [11]. Most of them use additional shared storage or third party software support to synchronize metadata modifications among the active and backup nodes. It introduces an extra overhead and affects normal metadata operations for system reliability. Metadata sharing and distributed storage have become key challenges for improving system scalability and reliability in distributed and cluster file systems.

In this study, we propose a novel Virtual Shared Storage Pool (VSSP) concept and design for metadata sharing and storage in HDFS. The VSSP is designed as a virtual storage device which is built on existing servers and transparent to upper layers. It provides common file interfaces such as read, write, and append for metadata storage. Different from traditional strategies, the VSSP uses a hybrid policy to distribute metadata according to different access patterns. For metadata modifications, we introduce a journal synchronization strategy by optimizing the two-phase commit protocol (2PC) [12]. It not only significantly reduces the overhead on metadata operations but also keeps state consistent among the server and backup nodes. To minimize the time of loading and storing namespace image, a fine-grained image replication approach is introduced. Based on the image partition, the metadata server can load namespace from the specified domain

978-1-4673-7891-8/15/$31.00 ©2015 IEEE



(a) Workflow of metadata storage

(b) Existing reliability mechanism based on metadata shared storage

Fig. 1: Metadata management and operations in HDFS

on demand. By distributing replicas with parallel pipeline transmission, it improves the I/O performance for namespace storage. Data checksum and integrity comparison are also used for metadata recovery. Experimental results show that VSSP improved the average performance by 40.51% and 23.46% when writing logs compared with the BookKeeper and Hadoop QJM (quorum journal manager) [7]. The average image read and write throughput was nearly 5 times and 2.4 times better than NFS [13] and the original pipeline approach in HDFS [4].

The rest of this paper is organized as follows. Section II discusses the background on metadata storage in HDFS. We then describe the detailed design of VSSP in Section III, including the metadata synchronization and replication strategy. Section IV presents experimental results. Section V provides a review of related work. Conclusions and possible future work are summarized in Section VI.

II. Background and Motivations

When providing metadata service, the namenode in HDFS stores metadata in memory for performance and on disk for persistence. HDFS uses the namespace state in memory to keep the organization of directories and files, which is saved as the fsimage file on the disk. For each client-initiated transaction, the metadata modifications, such as $create$, $delete$ and $mkdir$ operations, are recorded as the journal/edit and stored on the disk. A record of the image, called the checkpoint, is periodically written to the disk. As shown in Figure 1(a) and described below, there are several steps to store metadata for metadata operations in HDFS:

- Step 1: When the namenode receives an update, it writes the request to the journal and applies the request to in-memory data structure.
- Step 2: The namenode maintains the on-disk image of files and directories for application data. When the namenode reboots, it reads and merges the image and journal to reconstruct previous namespace state in memory. By merging them, a new on-disk image is created and saved in the disk.
- Step 3: By loading and saving image and journal files, the namenode can recover in-memory structure if necessary.
- Step 4: Periodically, a checkpoint is created on the disk for the namespace state. This process is often performed by a different host, such as the secondnamenode or checkpointnode. They retrieve the current image or checkpoint and journal files from the namenode, merge them locally, and return the new checkpoint back to the namenode.

In step 2 and 4, all log records for updates that have been reflected in the image or checkpoint can be truncated from the log. It avoids the continued increase of journal size. Current metadata storage can lead to missing or corruption of the journal and image files for local disk storage. Besides, it spends a long time for the secondnamenode or checkpointnode to retrieve metadata from the namenode through the network.

Figure 1(b) shows the current reliability mechanism [10] based on metadata shared storage in HDFS. The cluster has two metadata servers: an active namenode which provides metadata service and a backup node as a hot standby. Metadata transactions are written into the shared storage by the active and read by the standby simultaneously. In order to construct file locations, the datanodes report to both the active and standby. In the event of active crashes, the standby will take over its role and respond to the client. Based on the shared storage, the standby can synchronize metadata with the active and keep consistent state with it. However, current designs are often based on special shared storage or third party software for metadata synchronization and replication. It leads to additional overhead and metadata performance degradation.

The limitations of the current design as discussed above motivate us to devise a new metadata shared storage to achieve better scalability and reliability in HDFS.

III. Design and Implementation

In this section, we describe the detailed design and implementation of the proposed VSSP (Virtual Shared Storage Pool). VSSP uses a hybrid policy for metadata synchronization and replication which reduces the performance overhead on metadata operations and improves the I/O throughput for namespace storage.

A. Design of Virtual Shared Storage Pool

The VSSP is designed to store and share metadata in HDFS and to keep metadata persistent. The VSSP is built on existing cluster and needs no additional device or third-party software support. Each metadata server (MDS), including the namenode, standby node, and checkpointnode, is treated as a pool node for metadata storage. Datanodes can also be configured as pool nodes for redundancy. With a hybrid policy for metadata synchronization and replication, VSSP provides a virtual shared storage for metadata.

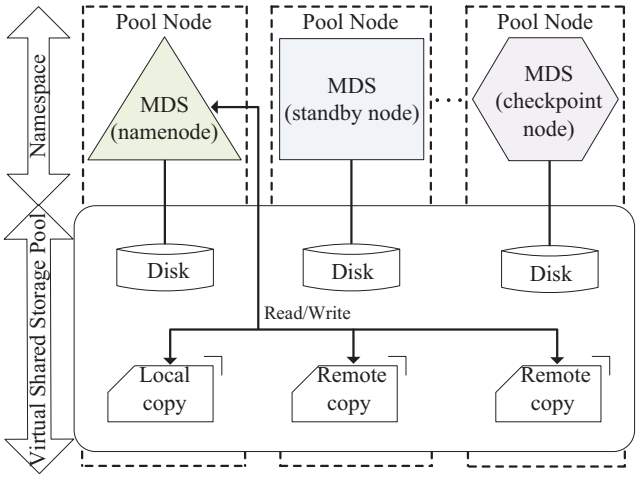


Fig. 2: Design of Virtual Shared Storage Pool (VSSP)

Figure 2 depicts the design of the Virtual Shared Storage Pool. As shown in the figure, VSSP consists of three pool nodes including a namenode, a standby node, and a checkpointnode. More servers can be added as pool nodes. VSSP provides common read/write interfaces for metadata persistence. Take the namenode for example, it writes logs to VSSP before performing metadata modifications. As the namenode is also a pool node, VSSP first flushes journals to its local disk when receiving the write request and then sends journals to other pool nodes through the network. For the standby node, it receives the journal streams, applies them to in memory structure and stores them in local disks. Combining synchronization with replication, the standby node can keep an up-to-date namespace state with the namenode in a reliable mechanism. For namespace state of namenode, the image is directly replicated in VSSP. When reading metadata, the MDS can benefit from VSSP with local file access. It improves the capability of loading metadata in HDFS. Based on VSSP, each MDS accesses journal and image files in a shared way. VSSP is transparent to MDSs in which they store metadata just like calling local interfaces. In order to further increase metadata redundancy, more replicas can be placed in the pool. The detailed synchronization and replication strategy will be introduced below.

B. Storage Directory

In our design, VSSP provides common file read/write interfaces for metadata persistence in HDFS. To ensure a transparent storage, VSSP uses a logical volume to maintain the mapping between metadata files and their locations. The volume is a tree-like storage directory that is maintained and kept with the same view in the MDS, which is also a pool node. When an MDS writes metadata, it submits the request to VSSP. For metadata persistence, VSSP selects appropriate pool nodes for storage. The metadata placement policy in VSSP provides a tradeoff between reducing the write cost, considering server roles, and increasing metadata reliability and read bandwidth. When metadata is written to journal or image files, VSSP places the first replica on the MDS where the writer is located. Other replicas are placed on remaining MDSs. If the number of replicas is greater than that of MDS, VSSP chooses the datanodes randomly as pool nodes for replication. Metadata is automatically restored when replicas are corrupted or unavailable. After the replica nodes are chosen, VSSP creates metadata files and directories on them via RPC calls.

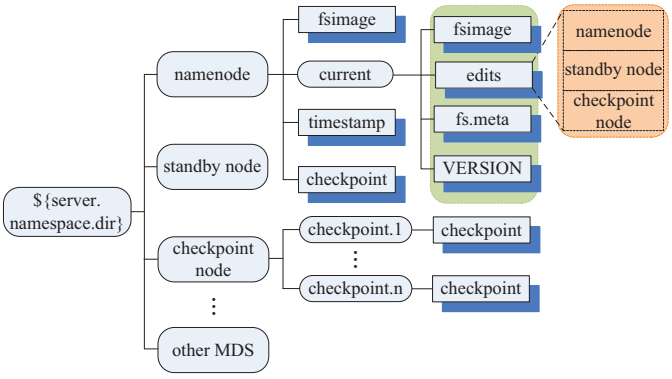


Fig. 3: Logical volume of the storage directory

Figure 3 illustrates the logical volume of storage directory. The tree root is the relative path which represents the metadata storage directory on local file system at each pool node. It has an MDS node list which keeps metadata persistence in HDFS. In each MDS node entry, it records metadata files and directories which are shown with shaded and rounded rectangles, respectively. For example, the *current* directory of namenode has four metadata files that include an image file (fsimage), a journal file (edits), a checksum file (fs.meta) and a version file. Extended attributes are added to each file entry and indicate which pool nodes its replicas belong to. In this case, the *edits* file of namenode is stored in three pool nodes, which are namenode, standby node, and checkpointnode.

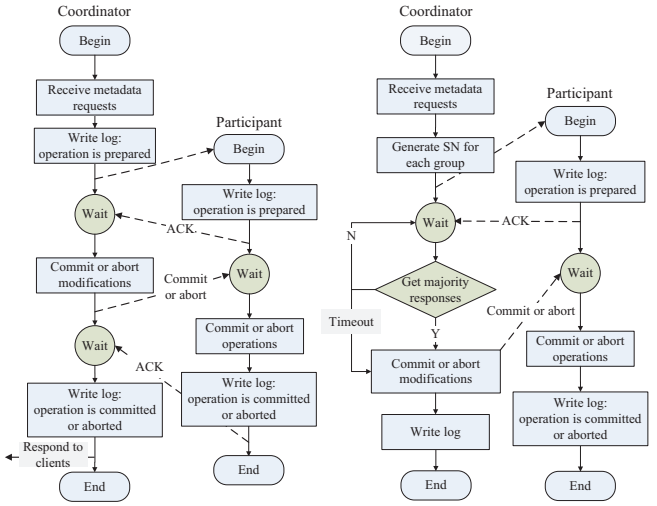
For flexibility, the logical volume is not persistent in storage devices. It is constructed by reporting from the pool nodes to the MDSs. When starting up, each pool node reads from the configuration or scans local disk to obtain the path of metadata files and directories. It sends the storage directory view to all MDSs. The pool node daemon in each MDS receives the

metadata location. It collects the storage directory view from all pool nodes, merges them, and constructs a logical volume in memory. Then the daemon compares checksums among metadata replicas and verifies the correctness for the same file. If two or more metadata replicas have the same value in checksum, it can be considered that they have the right information. As the journal or image files are always appended by one input stream, it can simply find the corrupted one by comparing file sizes. A $Recovery$ or $Remove$ operation will be sent to the pool node which has the obsolete replica. By this way, the pool node can repair the error file, create the missing directory, or re-replicate files from other pool nodes.

After constructing logical volume, the pool node daemon in the MDS selects an optimal replica from VSSP for metadata loading. The shared storage pool essentially acts as a virtual device that provides transparent metadata storage for MDS.

C. Journal Synchronization and Replication

When receiving client requests, the namenode records metadata modifications in the journal file before committing them. The namenode reconstructs in-memory structure by reading and replaying log records. Compared with local disk storage, the journal is distributed with multiple replicas in VSSP. In a highly reliable mechanism, MDSs need to synchronize metadtata to keep a consistent state among them, such as the namenode and standby node. To reduce the performance overhead on metadata operations, we design a distributed protocol based on the two-phase commit protocol (2PC) [12] for metadata synchronization and replication. The 2PC protocol is an atomic commitment protocol to achieve consistency for distributed operations. For comparison, we first review the 2PC algorithm and then introduce our protocol.



(a) Existing 2PC protocol

(b) Journal synchronization and replication protocol: optimized 2PC protocol

Fig. 4: Journal synchronization and replication in VSSP

Figure 4 shows the process of journal storage by using the optimized 2PC protocol in VSSP. Figure 4(a) shows the existing 2PC protocol method that includes the pool node who initiates the write operation (coordinator) and the pool node who receives log records (participant). The procedure is divided into two phases, a preparation phase and a commitment phase. In the preparation phase, the coordinator receives metadata requests from the client, writes the log with "Prepare" label in a local disk and transmits them to each participant. The participant receives journals and writes the "Prepare" log. In response, the participant sends "ACK" message to the coordinator to confirm the operation. The coordinator waits and collects replies from participants in the commitment phase. If receiving all "ACK" messages, it sends "Commit" messages to participants for execution. After all participants committed operations and write "Commit" logs, they send "ACK" messages to the coordinator for completion. When getting all "ACK" messages again, the coordinator returns results to the client. During the process, the operation will be aborted if any participant sends an "Abort" message or failures occur.

With the 2PC protocol for metadata synchronization, it can ensure metadata consistency among MDSs. However, it needs four times as the message communication between the coordinator and participant which will affect the metadata performance. Besides, the 2PC protocol lacks fault-tolerance mechanism in the commitment phase.

To address these issues, we design a new optimized 2PC protocol for metadata storage. Figure 4(b) describes the workflow of journal synchronization and replication in VSSP. First, the coordinator receives client requests and aggregates them in group. For each group, the coordinator assigns a monotonically increasing serial number (sn) for operation ordering and fault tolerance. It is described with the pair <sn, $groupid$>. When a group of log records is prepared, the coordinator sends it to all participants. Second, each participant receives metadata modifications, writes the "Prepare" log in local disk, and send an "ACK" message to the coordinator. At last, when the coordinator receives a majority of responses (more than half), it performs operations in memory and returns results to the client. Simultaneously, the coordinator notices the participant which have returned "ACK" message to commit and write logs. As the coordinator considers the synchronization operation has been finished, it deals with subsequent responses from participants via asynchronous threads. If the coordinator finds "ACK" messages timeout, it does not send journals to related participant any more. Different from the existing 2PC protocol, the coordinator concludes the operation success if more than half number of participants return "ACK" messages. During the process, the coordinator needs not write journals for two times. It stores the "Commit" or "Abort" logs in local disk directly. Supposing N is the number of replicas, our policy can tolerate $(N-1)/2$ failures and continue to work normally. As the journal file is written sequentially and read when the file system restarts or recovers, we relax the metadata consistency model for journal synchronization and replication. It does not maintain the same context for replicas at writing operations but ensures the eventual consistency when metadata reading operations happen.

In our policy, as soon as more than half number of "ACK" messages are returned, the coordinator responds to the client. Compared with the traditional 2PC protocol, it only waits for one "ACK" message from the participant and allows a small amount of replica failures. Although the strategy relaxes replica consistency, two ways are provided for eventual consistency guarantees and fault tolerance: one is by comparing journal file size or checksum to detect whether a replica has the up-to-date information. If the size of one replica is different with others or checksum errors occur, the pool node will retrieve missing log records according to the (sn) value from other pool nodes. The other method is to combine client with namenode operations to ensure metadata consistency. For service switching in the reliable paradigm, the new namenode can flush last group of logs to the standby node or the client may resend the failed operations for avoiding loss of metadata modifications. At the MDS side, duplicate journals are detected for metadata correctness. By providing metadata synchronization and replication in VSSP, our policy not only reduces the performance overhead on operations but also keeps state consistency between the namenode and standby node in HDFS.

D. Namespace Image Replication

The image is persistent in a file which represents the whole namespace state of metadata server in HDFS. It is often created when the namenode starts up or the checkpointnode makes a checkpoint. As the size of image file is much larger than that of the journals, we perform the replication through the pipeline pattern.

In our approach, the image is divided into multiple partitions and replicated concurrently. As HDFS is mainly designed for MapReduce jobs in which applications access files or directories with the full path, we split the namespace into different images according to the pathname. With this idea, the partition algorithm is described in algorithm 1.

Algorithm 1 Image partition algorithm in VSSP

```
1: f = the full pathname of the file or directory
2: Dir Function = the prefix path of the file
3: ptrnum = the number of image partition
4: stream[ptrnum] = streams of image partition
5: for each file or directory ∈ namespace do
6:    if file then
7:       i ⇐ Hash(Dir(f)) mod ptrnum
8:       stream[i] ⇐ file
9:    else
10:      j ⇐ Hash(f) mod ptrnum
11:      stream[j] ⇐ directory
12:   end if
13: end for
```

Based on the partition algorithm, the namespace of HDFS is divided into fine-grained images and distributed in multiple pool nodes. Benefiting from logical volume report, the MDS knows which image partition should be read when constructing the namespace state. It means the namenode can load namespace from the specified domain on demand.

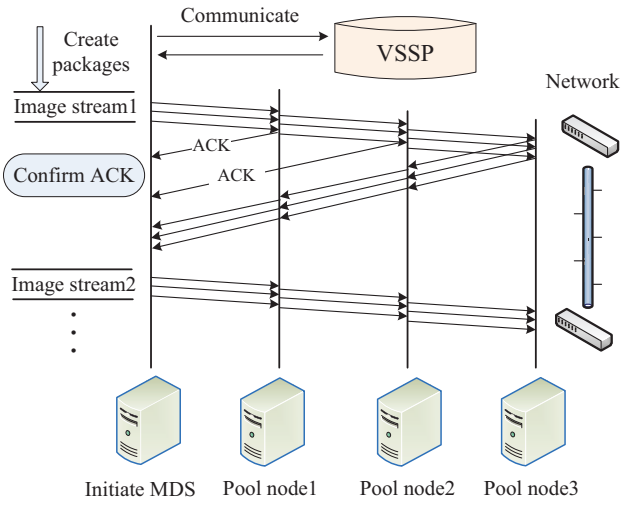


Fig. 5: Image replication in VSSP

After partition, the images are replicated and transmitted in a pipeline fashion. Different from the existing pipeline approach in HDFS [4], VSSP transmits data concurrently and concludes the success when more than half the number of replicas return confirmation messages. Figure 5 shows the workflow of image replication in VSSP. When the MDS stores namespace image, it treats each image partition as a stream and sends them simultaneously. Supposing the image has a replication factor of three, the MDS first chooses appropriate pool nodes with proposed metadata placement policy or retrieves a list of pool nodes from the logical volume. Then the MDS sends each image stream to the first pool node by creating data packages. The first pool node receives packages, writes them to its local disk and sends them to the second pool node. Meanwhile, the first pool return an "ACK" message to the MDS immediately. The second pool node repeats the same operation and forwards the packages to the next one. At last, the final pool node writes the data packages and sends a reply to the previous one for confirmation. In the traditional pipeline approach, the data replication is considered successful until the reply of the final pool node is delivered to the MDS. In our design, pool nodes in the first half of the replica node list will return "ACK" messages to the MDS in advance. Once receiving more than half replica number of "ACK" messages, the MDS concludes the replication success. The remaining returned messages will be received by the asynchronous thread in the MDS. Algorithm 2 describes the image replication algorithm in VSSP. This design reduces the waiting latency for messages and improves the write performance for image replication.

IV. Evaluation

In this section we present the evaluation results of the proposed Virtual Shared Storage Pool (VSSP). The tests were performed from three aspects. We first measured the performance of VSSP for journaling compared with classical

Algorithm 2 Image replication algorithm in VSSP

```
repnum = the number of replica factor
targets[repnum] = the list of replica pool nodes
acknum = the number of ACK messages
for each stream ∈ namespace image do
    //send the stream to target nodes
    for each packet ∈ stream do
        success = false
        //send the packet to target nodes
        while notsuccess do
            for each i ∈ [0, repnum] do
                if targets[i] return ACK then
                    acknum++
                end if
                if acknum ≥ repnum/2 then
                    acknum++
                    return success
                end if
            end for
        end while
    end for
end for
```

policies. Then we tested the read and write throughput for image storage. At last we tested the ability of metadata restoration in case of failures, which validates the metadata consistency guarantee in our design.

The experiment platform is a 20-node cluster. Each node consists of four Intel Xeon X3320 Processors, 8-GB memory, and one Gigabit network interface card, with Linux kernel 2.6.32. Each of them acts as the pool node in VSSP and stores journal and image files. For file system operations, multiple metadata modifications are aggregated before being submitted and written back to the journal in an asynchronous way.

A. Evaluation of Journaling

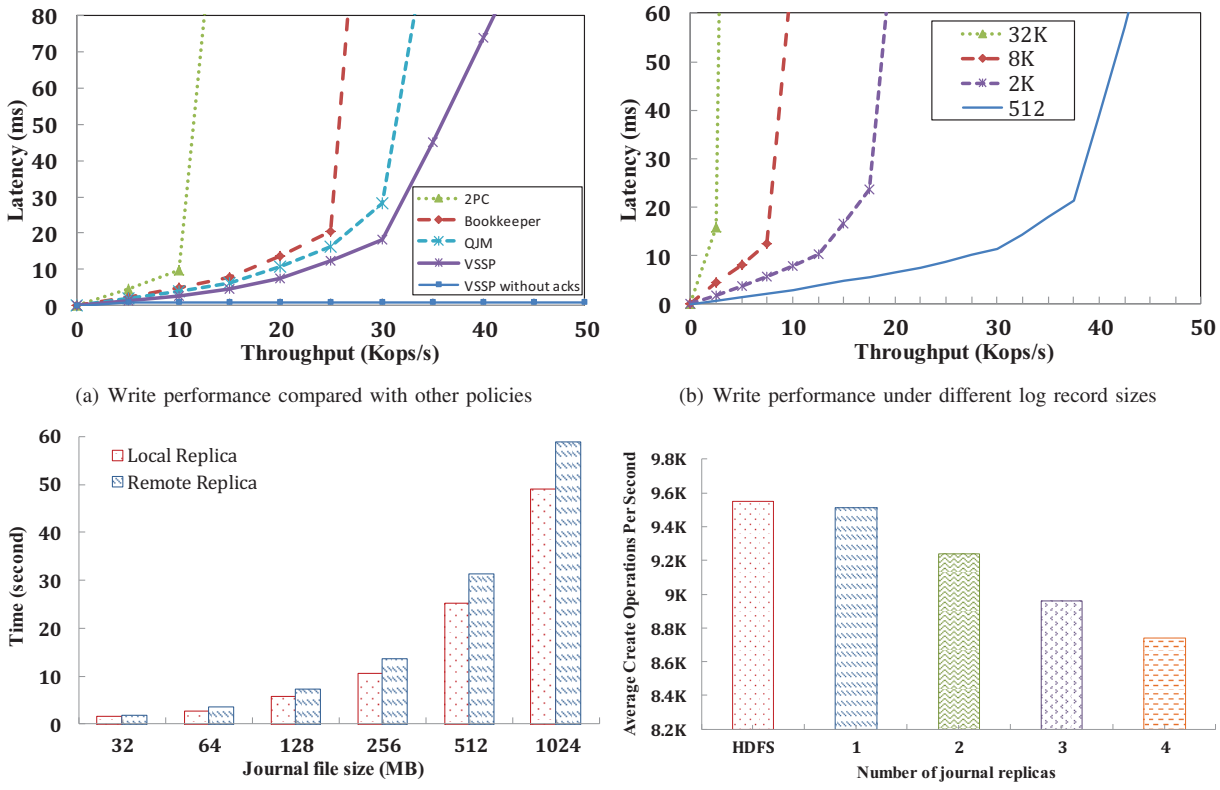
When responding to client quests, the namenode records logs for metadata modifications. In our design, the namenode stores journals in the VSSP. To compare VSSP with other approaches, typical policies, including 2PC, Hadoop Quorum Journal Manager (QJM) [7], and BookKeeper log system [14], were implemented or deployed for metadata synchronization and replication. The tests used one node as the log writer and placed three replicas for journals.

Figure 6(a) compares the journal write performance under different policies. The x-axis represents write counts per second and the y-axis represents the average delay latency. All logs were flushed asynchronously with 1KB size. From the results shown in the figure, it can be observed that the performance of 2PC was the worst among all policies. Its latency was nearly 2, 3, and 4 times of the values of the BookKeeper, QJM, and VSSP, respectively. This is because the journal replication in the 2PC needs four round trip messages, which considerably increases the waiting time and load of nodes. The BookKeeper is a logging storage system and writes replicas to assigned redundant nodes simultaneously. However, in order to ensure strong consistence, BookKeeper waits for an operation to be completed until all replicas have been written successfully. Otherwise, it selects other nodes to rewrite journals via a round-robin algorithm when replica errors occur. The QJM is a log sharing storage used between the active and standby namenodes. It can tolerate at most $(N-1)/2$ failures but needs additional overhead for journal synchronization and lacks the mechanism of fault tolerance. Compared with the BookKeeper and QJM, VSSP shows a better result, which improved the performance by 40.51% and 23.46%, respectively when writing 25K logs per second. This is because VSSP adopts a replication policy that combines local files and remote copies. It reduces the communication overhead in which VSSP returns success only if more than half replicas are written. All write latency increased gradually with the increase of the throughput except in the case of writing without responses (VSSP without acks). It is due to the processing capability limit of the writer node.

Figure 6(b) reports the write performance under different log sizes in VSSP. The results show that VSSP achieved better performance with less size logs. For example, the average write latency has been increased nearly 20 times when the log size varied from 512B to 32KB. In HDFS, journal storage belongs to a small chunk of writing data operations. It well verified the effectiveness of VSSP for metadata synchronization and replication.

Log records are stored in the journal file with multiple replicas in VSSP. We have also performed tests to measure the read performance of VSSP by using the namenode in HDFS to read journals. Figure 6(c) reports the time spent for reading journals with different sizes and replicas. The results reveal that the read performance of local journal for namenode was about 22.32MB/s, which is similar with that of HDFS. The time increased with the increase of journal sizes. As the namenode would replay logs to construct namespace in memory, it spent additional overhead for reading journals. When the namenode read logs from remote replicas, there is a performance degradation which was reduced by 19.59% on average compared with local copies. Network communication caused delays, but the effect is negligible. The VSSP still achieved impressive read performance with the replication strategy of increasing metadata availability.

We have conducted another series of tests to measure the influence on metadata operations with the journal write policy in VSSP. These tests used the namenode to provide metadata service and multiple clients on different nodes for payload. Figure 6(d) shows the average create operations per second with different journal replicas. It can be observed that metadata operations with one local journal achieved the best performance, which is nearly equal to that of HDFS. With the increase of replica number, the average create operations per second were reduced. This is because the namenode would synchronize and replicate journals through VSSP for multiple replicas, which incurs additional data transmission overhead. The create performance was decreased by 2.86%, 2.99%, and 2.45%, respectively, when adding one replica each time. The results confirm that the VSSP achieved metadata synchronization and replication with little effect on the performance.



(a) Write performance compared with other policies

(b) Write performance under different log record sizes

(c) Read performance with different journal sizes

(d) Performance of create operations with different replica numbers

Fig. 6: Performance of journal storage in VSSP

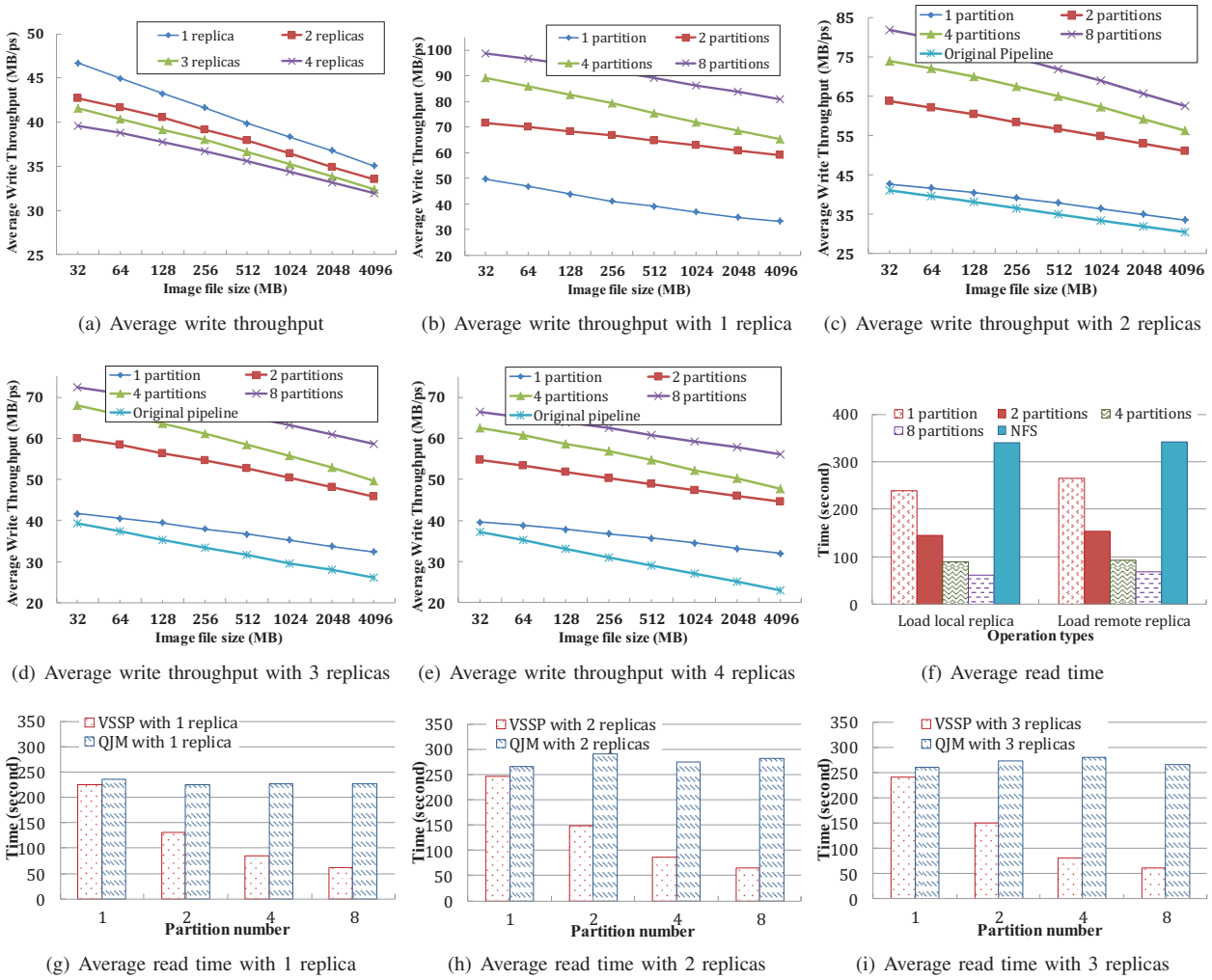
B. Evaluation of Image Storage

When the namenode starts up or the checkpointnode makes a checkpoint, they write namespace state to the image file in VSSP. Different from journals, image storage belongs to large-size file write operations. Our VSSP achieves a fine-grained image replication with parallel pipelined transmission. The tests focused on the average read and write performance of image in VSSP. For comparison, we implemented an original pipeline approach like data replication in HDFS [4] and chose NFS [13] as the shared storage to validate the effectiveness of the VSSP replication strategy. We also used the Hadoop QJM [7] for image read test under different replicas. The image file was generated with various sizes by adopting multiple clients to produce payload. Table I summarizes the test payload and resulting image files in which the image with longer size represents larger file system scale.

TABLE I: The test payload and resulting image files

Journal Size (MB)	Image Size (MB)	Log Record Number (K)	Metadata Operation Number (K)
135.72	43.64	521	1281
268.41	86.93	1031	2763
576.28	170.37	1991	5129
1521.56	486.91	3483	10537
3145.12	1106.45	6988	19770
5790.43	2082.35	13049	35976
10237.72	4260.94	23878	68117

The test first measured the average write throughput by writing image files with different sizes. Each file was written ten times and the average value was calculated. For replication, the packet size was set to 64KB and the chunk size was set to 512B. The results are shown from Figure 7(a) to Figure 7(e). Figure 7(a) shows that the write performance of VSSP tends to reduce and stabilize with the increase of image size and replica number. This is because VSSP divides the image into packets and confirms the replica to be written, which needs additional waiting time. Compared with the disk I/O, the throughput declined for image write. As the namenode must parse the namespace structure and write it to a defined format file, it incurs an overhead for image storage. For each replica added in the VSSP, the average write throughput was reduced by 6.05%, 3.12%, and 3.14%, respectively. It proves that there was little influence on the performance with the VSSP replication policy. Figure 7(b)-7(e) report the results of the original pipeline approach in HDFS and our approach with different replicas and partitions. As expected, the performance gain increased when the image was divided into multiple partitions with different directories. Due to the directory-based partition and concurrent writes, the average performance of VSSP with one replica was improved by 61.08%, 89.95%,



(a) Average write throughput (b) Average write throughput with 1 replica (c) Average write throughput with 2 replicas

(d) Average write throughput with 3 replicas (e) Average write throughput with 4 replicas (f) Average read time

(g) Average read time with 1 replica (h) Average read time with 2 replicas (i) Average read time with 3 replicas

Fig. 7: Performance of image storage in VSSP

and 121.58%, when doubling partition numbers each time. With the addition of replicas, the write throughput degraded due to data copying and network communication. Compared with the original pipeline approach in HDFS, our approach achieved more than 2.4 times of performance improvement when placing 4 replicas with 8 partitions. These test results confirm that our approach significantly improved the write performance of images.

We have also measured the read performance by using the namenode to load a 5GB image file. The NFS and Hadoop QJM are regarded as shared storage to provide metadata access. Experimental results are shown from Figure 7(f) to Figure 7(i). In Figure 7(f), we can observe that the average read time spent of VSSP was less than NFS, either from local file or from remote replica. As the namenode needs to mount the NFS file system and writes image to separate shared storage, it leads to additional costs. Benefiting from concurrent read operations, the performance of VSSP with 8 partitions was nearly 5 times better than NFS. Hadoop QJM is mainly designed to share edit logs between the active and standby namenode for HDFS. We used it as a distributed storage to test read performance. Figure 7(g), Figure 7(h), and Figure 7(i) show that the performance was close between the VSSP without partitions (1 partition) and Hadoop QJM. This is because the image can be loaded from local disk copy in both storage. When dividing the image into multiple partitions, the VSSP achieved better read performance than Hadoop QJM. As our approach can retrieve the image replicas and construct the namespace state from different partitions concurrently, it reduces the average read time spent for image. The read performance of VSSP gained an improvement of more than 430% when the partition number was 8 and the replica number was 3.

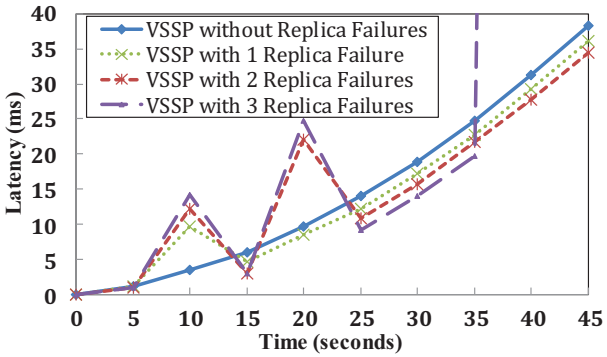
In conclusion, image storage with the replication strategy in VSSP not only enhances the read/write throughput but also improves the metadata availability.

C. Evaluation of Metadata File Restoration

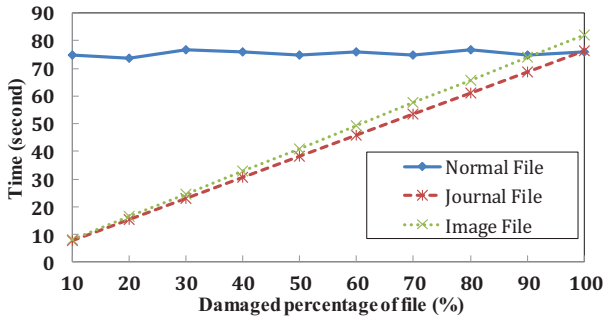
To verify metadata restoration in SSP, a series of tests were performed for metadata writes and file recovery in case of failures. The VSSP was configured with 4 pool nodes which are also nodes of placing replicas. Figure 8 plots the journal

write performance when different replica errors occurred (the vertical axis is the average access latency). In the tests, we increased the workload gradually and generated errors by killing pool node processes. When there was one replica error in the 10th seconds, the latency increased as much as three times for failure free cases. This is because the VSSP waits for the response from the failed pool node and removes it when timeout, which incurs an additional overhead. But the VSSP was recovered to normal performance subsequently. With our journal replication approach, the VSSP returns success when more than half numbers of replicas are written. Although the written replicas are decreased, the performance of VSSP is improved with the reduction of response time and waiting latency. The experiments show the same results when there were two replica faults. In the case of more than three replica errors, the journal writes would fail because the number of written replicas was less than half. It resulted in the linear increase of the average latency. Figure 8(b) shows the recovery time for different damaged files with each 5GB total size. The curve on the top represents the common method of repairing a normal file. It retransmits the whole file and spends more time regardless of the failure reasons. In our design, the VSSP reports metadata file locations to construct logical volume on all MDSs when starting up. As the journal or image file belong to sequential write, the MDS can detect replica errors by comparing file size when their checksum are the same. Due to the method of resuming transmission from fault points, the damaged file can be recovered in a proper position. The test results reveal that it took shorter time for metadata file recovery when there was smaller error proportion.

In conclusion, the experiments show that the VSSP can achieve impressive write performance and short recovery time in the case of replica failures. It confirms the effectiveness and reliability of the VSSP metadata replication strategy as well.



(a) Journal write performance with different replica errors



(b) Recovery time for different damaged files

Fig. 8: Metadata restoration in VSSP

V. Related Work

HDFS has been widely used for mass data storage and management in Hadoop cluster. Although many studies have been conducted to improve various characteristics of HDFS, there is no effort in literature, to our knowledge, to optimize metadata storage with a virtual shared storage pool like what we have introduced in this research.

Many parallel/distributed file systems, such as Lustre [15], StorageTank [16], GFS [17], and HDFS [4], ensure metadata persistence and consistency by logging metadata updates. It helps to reconstruct namespace state but the metadata may be lost partially or entirely due to disk or volume failures.

Distributed metadata storage is an effective way to improve metadata availability. Ceph [18] uses an object storage cluster to keep in-memory file and directory structures. HDFS Federation [7] configures multiple storage directories or the remote NFS [13] server to store the journal and image. Bigtable [19] uses a three-level tree hierarchy to organize tablet locations and store the metadata information in HDFS. These approaches prevent metadata loss from local disk or single node failure, but they lack a reliable mechanism for metadata redundancy or need additional device or third-party software support.

There are literatures related to metadata service availability that mainly focus on synchronization strategy to keep consistent states among servers. In the BackupNode [4], the primary node directly sends journals to the standby node. It does not affect metadata operations but cannot ensure metadata consistency. Yahoo! combines BackupNode with Linux shared device DRBD [20] to achieve active-standby switching. Hadoop HA Branch [7] uses NFS [13] or the quorum journal manager (QJM) [7] for synchronization. S2PC-MP [21] combines the 2PC protocol with metadata processing to perform cross-metadata server operations. Although above approaches ensure metadata state consistency, they can lead to additional overhead on normal metadata operations.

A number of other research studies have been conducted for replication in metadata storage. ZooKeeper [22] is a centralized service for maintaining configuration information, providing distributed synchronization, and providing group services. It can provide effective and reliability service for metadata replication but will result in additional overhead. Bookeeper [14] is a distributed logging storage. It writes the journal stream into multiple nodes for replication. Bookeeper achieves effective storage for journals but needs to rewrite the data when any replica fails. When performing replication, the Paxos algorithm [23], two-phase commit protocol (2PC) [12] and similar ideas are often adopted to ensure replica consistence. PacificA [24] advocates a general replication framework

for commonly used log-based storage systems. It uses the primary-backup strategy for data replication. Lease mechanism is taken from the primary to detect secondary failures and reconfiguration. HARP [25] and Echo [26] adopt a variation of the primary copy to replicate logs on all secondaries. These schemes provide high availability for metadata replication, but they focus on the strong consistence while reducing the metadata performance. Other mechanisms enable multiple-pipeline data transfer [27] or rdma-based method [28] to improve replication performance. However, they are mainly designed for data placement and are not for metadata storage.

VI. Conclusion

Motivated by the increasing needs of mass data storage in Hadoop cluster, we introduce a novel Virtual Shared Storage Pool (VSSP) concept and design for metadata storage in HDFS in this study. Different from traditional shared storage, the VSSP uses a hybrid policy to distribute metadata. Combining journal synchronization based on an optimized 2PC protocol with fine-grained image replication, the VSSP provides transparent storage for the metadata server in HDFS. We have conducted extensive tests to verify and evaluate the proposed VSSP. Experiment results show VSSP improved the average performance by 40.51% and 23.46% when writing logs compared with BookKeeper and Hadoop QJM. The average image read and write throughput was nearly 5 times and 2.4 times better than NFS and the original pipeline approach in HDFS. These tests confirm that VSSP enhances the metadata scalability, availability, and access performance for HDFS.

In the future, we plan to investigate VSSP's impact on upper applications, including performing standard MapReduce jobs and tasks, measuring performance impact or recovery time on the node failures and etc. We also plan to improve VSSP functionality on different storage platforms such as the heterogeneous devices with HDD and SSD.

Acknowledgment

This work is supported by the National High-Tech Research and Development Program of China under grant 2013AA013204, supported by the National HeGaoJi Key Project under grant 2013ZX01039-002-001-001, and by the Strategic Priority Research Program of the Chinese Academy of Sciences under grant XDA06030200, and in part by the National Science Foundation under grant CNS-1338078 and IIP-1362134 through the Nimboxx membership contribution.

References

[1] V. Turner, J. F. Gantz, D. Reinsel, and S. Minton, "The digital universe of opportunities: Rich data and the increasing value of the Internet of Things," White Paper, IDC_1672, International Data Corporation, 2014.

[2] T. Hey, S. Tansley, and K. Tolle, "The fourth paradigm: data-intensive scientific discovery," Microsoft Research, Microsoft Corporation, 2009.

[3] T. White, "Hadoop: The definitive guide," O'Reilly Media, 2012.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *26th IEEE Transactions on Computing Symposium on Mass Storage Systems and Technologies*, Incline Village, USA, May 2010, pp. 1–10.

[5] K. V. Shvachko, "HDFS scalability: The limits to growth," *login*, vol. 35, no. 2, pp. 6–16, 2010.

[6] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron, "Scale-up vs scale-out for Hadoop: Time to rethink?" in *Proc. of the the 4th annual Symposium on Cloud Computing, (SOCC'03)*, Oct. 2013.

[7] "Hdfs federation," 2014. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html.

[8] Y. Wang, J. Zhou, C. Ma, W. Wang, D. Meng, and J. kei, "Clover: A distributed file system of expandable metadata service derived from hdfs," in *International Conference on Cluster Computing*, 2012, pp. 126–134.

[9] I. P. Egwutuoha, D. Levy, B. Selic, and S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems," *J. of Supercomputing*, vol. 65, no. 3, pp. 1302–1326, 2013.

[10] D. Borthakur, J. S. Sarma, J. Gray, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer, "Apache Hadoop goes realtime at Facebook," in *Proc. of the International Conference on Management of data (SIGMOD)*, Athens, Greece, Jun. 2011, pp. 1071–1080.

[11] J. Lin, F. Leu, and Y. Chen, "ReHRS: A hybrid redundant system for improving MapReduce reliability and availability," *Modeling and Processing for Next-Generation Big-Data Technologies*, vol. 4, pp. 187–209, 2015.

[12] M. T. Ozsu and P. Valduriez, "Principles of distributed database systems," 3rd Edition, Tech. Rep., 2011.

[13] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the Sun network filesystem," in *Proc. of the Summer USENIX conference*, Portland, USA, Jun. 1985, pp. 119–130.

[14] "Apache BookKeeper," 2010. [Online]. Available: http://zookeeper.apache.org/bookkeeper.

[15] P. J. Braam, "The Lustre storage architecture," White Paper, Cluster File System, Inc., Oct. 2003.

[16] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg, "IBM Storage Tank-a heterogeneous scalable SAN file system," vol. 42, no. 2, 2003, pp. 250–267.

[17] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," in *Proc. of the 19th Symposium on Operating Systems Principles, (SOSP '03)*, New York, USA, Oct. 2003, pp. 29–43.

[18] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. of the 7th Symposium on Operating Systems Design and Implementation*, 2006, pp. 307–320.

[19] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *the 7th symposium on Operation Systems Design and Implementation*, Incline Village, USA, 2006, pp. 205–218.

[20] F. Haas, P. Reisner, and L. Ellenberg, "The DRBD User's Guide," LINBIT Information Technologies GmbH, 2009.

[21] J. Xiong, Y. Hu, G. Li, R. Tang, and Z. Fan, "Metadata distribution and consistency techniques for large-scale cluster file systems," vol. 22, no. 5, 2011, pp. 803–816.

[22] "Apache ZooKeeper," 2010. [Online]. Available: https://zookeeper.apache.org

[23] L. Lamport, "The part-time parliament," Research Report 49, Tech. Rep., 1989.

[24] W. Lin, M. Yang, L. Zhang, and L. Zhou, "Pacifica: Replication in log-based distributed storage systems," Technical Report MSR-TR-2008-25, Microsoft Research, 2008.

[25] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shrira, and M. Williams, "Replication in the Harp file system," in *13th Symposium on Operating Systems Principles*, California, USA, 1991, pp. 226–238.

[26] G. Swart, A. Birrell, A. Hisgen, and T. Mann, "Availability in the Echo file system," Research Report, Systems Research Center, Digital Equipment Corporation, 1993.

[27] H. Zhang, L. Wang, and H. Huang, "SMARTH: enabling multi-pipeline data transfer in HDFS," in *In the 43th International Conference on Parallel Processing*, 2014, pp. 30–39.

[28] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, "High performance rdma-based design of hdfs over infiniband," in *High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–12.