HDFS Space Consolidation

Aastha Mehta*[,1,2], Deepti Banka*[,1,2], Kartheek Muthyala*[,1,2], Priya Sehgal[1], Ajay Bakre[1]

*Student Authors

[1]Advanced Technology Group, NetApp Inc., Bangalore, India

[2]Birla Institute of Technology and Science, Pilani, India

Abstract

Recently, many corporate organizations have started using private Hadoop clusters to perform their tasks. Each of the Hadoop cluster nodes consist of its local computing and storage resources. There are many instances in such corporate clusters where significant amount of disk capacity on each of these cluster nodes remains unused. In this work, we consolidate the total idle disk space in a Hadoop cluster and present it to the client as an iSCSI LUN, which acts like a reliable store. The client, who is outside the Hadoop cluster, can store data in this LUN via an iSCSI initiator. We leverage Hadoop's distributed file system (HDFS) at the backend to distribute and maintain appropriate replicas of data existing within the LUN and recover from node or disk failure through replica regeneration. Thus, we achieve reliability of client data over the consolidated HDFS space.

1. Introduction

There is an increasing popularity of open source data management solutions using technologies like Hadoop [2,3,5]. Hadoop clusters typically employ commodity servers with large amount of direct attached storage (DAS) that can be used for distributed computing tasks using the MapReduce paradigm. HDFS is used to store the data required by MapReduce jobs and to store its results, which are replicated on the cluster nodes for resiliency. Since DAS capacities have gone up to TBs per node in today's commodity servers, it is possible that a significant storage capacity remains unused in the cluster. Currently it is not possible to use the space available in Hadoop cluster from outside the cluster because of two limitations of the Hadoop Distributed File System (HDFS) [6]:

- HDFS does not consolidate the whole space available in the cluster.
- HDFS does not support over-writes.

In this work, we address these two problems and allow access to the spare disk capacity within the cluster for tasks, which reside outside the cluster. The spare disk space can be consolidated and exported as an iSCSI LUN to the client. At the Hadoop backend, the data written to the LUN can be managed in the form of HDFS files. Reliability can be achieved by replicating the HDFS files appropriately on the data nodes of the cluster.

This work is similar to MapR [13] Direct Access NFS that allows access to the HDFS namespace through NFS clients. The only difference between MapR and our work is that we export the HDFS space through iSCSI, while they export it through NFS. The main contributions of this paper are –

- We designed the solution architecture for consolidation in distributed systems
- We implemented a translation layer on top of HDFS, to enable client operations in the consolidated space

Rest of the paper is organized as follows. Section 2 provides details about the previous work and explains the key concepts. Section 3 explains the design of our solution architecture and section 4

[2]Corresponding authors – undergraduate students at BITS Pilani, carried out work as interns at NetApp, Inc., Bangalore

discusses about the performance of our system. We conclude in Section 5 with future work plan.

2. Background

In our previous approach, Usage of Idle Space in desktops and laptops [1], consolidated storage space was exported from the hard disks of individual workstations like user laptops. The iSCSI target software was installed on each of these laptops with the idle storage. The iSCSI initiator on a client discovered the iSCSI targets on the laptops and presented them to the client as SCSI disks. The volume manager at the client end then configured RAID over these independent disks. There are certain limitations of this approach.

- This approach moves the onus of consolidation to the client end software, like a volume manager.
- It also depends upon the client to take care of any data loss by having a RAID solution over these discovered LUNs. If a computer goes offline, the RAID reconstructs the disk from the spare pool. But this does not guarantee data availability at all the times because there can always be a window in which there might be no spares available with the system and the raid group just fails.

Sudarshan et. Al [12] worked on aggregating unused space using their Freeloader architecture. This architecture primarily consisted of three types of nodes:

- Benefactor nodes: These nodes donate the idle space from their local storage.
- Master nodes: These nodes manage the metadata of the benefactor nodes.
- Client nodes: Applications running on these nodes accessed the storage provided by the benefactor nodes.

This architecture exposes API's like put(), get(), FL_open(), FL_write(), FL_read(), FL_close() etc.. to allow access to the data. POSIX compliant applications cannot be readily deployed in this environment because they are limited to use APIs that freeloader provides.

Our work is similar to Kartheek, et.al [1] and Sudarshan et al[12], as we consolidate the space amongst a bunch of nodes, but the onus of reliability and space consolidation resides with the space provider, which in our case is the Hadoop Distributed File System (HDFS) [6] and not shifted to the client (iSCSI initiator). And the supportability to posix compliance is provided by the inbuilt fuse-dfs component of Hadoop.

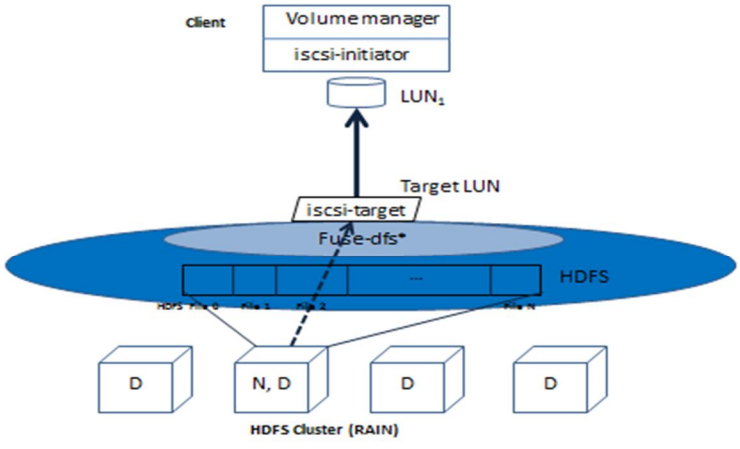

Figure1: Design architecture

Hadoop [2,3,5] is an Apache Project and it provides a distributed file system for the analysis and transformation of very large datasets using MapReduce [4] paradigm. HDFS is the file system component of the Hadoop. It works at a layer above the local file system and exposes HDFS API's to applications. HDFS stores file system metadata and application data separately. It stores the file system metadata on a dedicated server called *namenode* like most of the distributed file systems like PVFS[7][8], Lustre[9] and GFS[10]. All the application related data is stored on data nodes. In contrast to Kartheek's [1] solution, reliability overhead now resides completely with the name node. HDFS protects data by maintaining replicas of same data on different data nodes. We leverage the reliability feature inherent in HDFS to provide a reliable consolidated space to the client.

3. Design

Figure 1 shows the architecture of our work. It consists of a Hadoop cluster of *x* nodes with HDFS. One node is a namenode, denoted by N, and all the nodes are datanodes, denoted by D. HDFS is not POSIX compliant. Therefore, a FUSE file system, called Fuse-DFS [11] is implemented, that mounts HDFS and translates POSIX file system calls to HDFS calls and vice-versa. As shown in the figure, we use a modified version of Fuse-DFS on one of the nodes (mostly namenode), to consolidate the entire space available on the Hadoop cluster and export it as a single large file. The iSCSI target software installed on the namenode exports this large Fuse/HDFS file as a LUN to the client.

The client has iSCSI initiator software installed on it. The initiator discovers the LUN exported by the target and presents it as a block device. The client can create a file system directly on this block device. Alternately, it can ask the volume manager to create software RAID with other discovered LUNs, and then create a file system on that volume.

To achieve this high level architecture, we had to resolve two main problems. Firstly, the basic task was to consolidate the space available in HDFS cluster and expose it as a big block device. Secondly, the initiator's volume manager requires creating a file system on top of this consolidated space. The client operations will involve

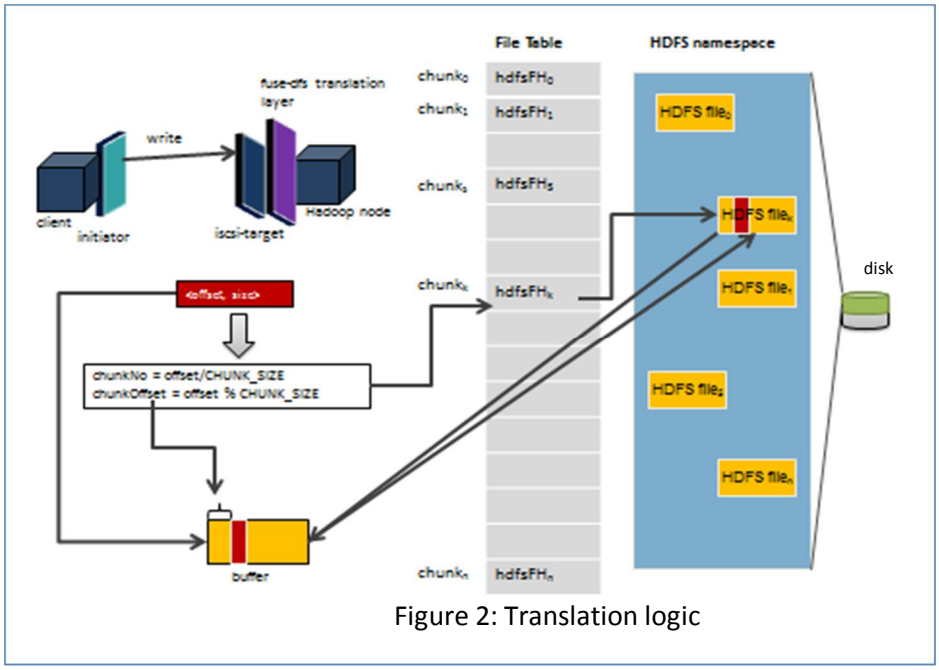

Figure 2: Translation logic

modifications to its already created files. However, HDFS does not allow appends or over-writes to existing files [14] in its namespace. Hence, we need to provide support for over-writes in a different layer than HDFS.

To address these problems we introduced translation logic in Fuse-DFS. For the rest of the document, we call the modified Fuse-DFS as Fuse-DFS*. When Fuse-DFS* is first started, it identifies the LUN names to expose by parsing the ietd.conf file. In our work, one iSCSI target present on one node exposes only one LUN. Fuse-DFS* consolidates the space and makes it available as this LUN name. Next, iSCSI target starts with the LUN name and queries Fuse-DFS* about the size of the LUN. Fuse-DFS* returns the size of the entire space available on HDFS, although there is no physical file with the LUN name present in HDFS. This modification helps to consolidate the space.

To allow over-writes in HDFS without changing HDFS itself, we modified Fuse-DFS. The iSCSI LUN, i.e. the consolidated space, is internally broken into fixed size chunks, called LUN chunk files, by Fuse-DFS*. The size of the chunk file is independent of the size of iSCSI data blocks, and is a configurable parameter. Typically it should be a multiple of the HDFS block size which is the basic unit of allocation in HDFS. Bigger chunk size enables transfer of more amount of data in one operation, thus minimizing the effect of disk seek latencies. On the other hand, for every change of even 1 byte to a chunk the entire chunk has to be modified. In this case, a larger chunk size will increase the write overhead. We are currently experimenting with chunk size less than HDFS block size and different multiples of HDFS block size.

The translation logic maps iSCSI data blocks to chunk files, which are then mapped to HDFS files. It maintains the metadata for all chunk files in memory in a table, called file table, which is indexed by the chunk number. The metadata contains a handle for the HDFS file corresponding to the chunk file along with other information like file size, open mode, etc.

The client operations are sent by the initiator to iSCSI target in form of CDBs (command data blocks). The command received at iSCSI target contains the SCSI request in the form of <*op, offset, size*>. Here *op* defines read or write operation, *offset* defines the start offset into the LUN for the *op*, and *size* defines the number of consecutive bytes to be read or written from *offset* into the LUN. The translation logic indexes into the file table using *offset*. If the *op* is a write and if there is no file handle in the indexed chunk record, the request is to write new data in the chunk file i.e. create a new corresponding HDFS file. Fuse-DFS* calls create function to handle the request. If the write request indexes into a chunk record with an existing file handle, it indicates an over-write request. In this case, Fuse-DFS* first accesses the file using the file handle from file table, and reads its contents into a buffer in memory. It performs the over-write into this buffer and then calls write to create a new HDFS file. It deletes the old version of the file and updates the metadata in the file table for new file, and the file handle to point to the new file.

4. Performance

We compared the performance of Fuse-DFS* including our translation logic with the performance of unmodified Fuse-DFS. We configured a one node Hadoop cluster formatted with HDFS. The node thus acts as both namenode and datanode. In one case, we installed Fuse-DFS* on top of HDFS and in another, we installed unmodified Fuse-DFS on top of HDFS. We used simple micro benchmarks for read and write operations on both setups and recorded latency and throughput of the setups. We performed the I/O for chunk size of 4096 bytes and file sizes ranging from 4096 bytes to 2GB. We have not involved

iSCSI target and initiator software layers in our experimentation. This is because the control experiment consisting of unmodified Fuse-DFS cannot consolidate the disk space under HDFS and export via iSCSI.

Our translation adds significant overhead to the original Fuse-DFS code. This is because we are using a sub-optimal chunk size of 4096 bytes while HDFS block size is 64MB. However, we are currently working to improve the performance of our entire system including the Fuse-DFS*, the iSCSI target and initiator software layers as a whole. We are experimenting with various sizes for chunk files to find a size optimal for acceptable I/O throughput and latency.

5. Conclusion

We have successfully addressed the problem of consolidating space by leveraging HDFS architecture and by solving the problem of read-modify-write with our fuse translation layer.

In future, we plan to experiment with different chunk sizes and compare its impact on both read and write performance. We also plan to improve the performance of the total system including the iSCSI layers.

References

[1]. Kartheek Muthyala, Ajay Bakre. Usage of Idle Space in Desktops and Laptops. Poster session presented at: Grace Hopper Conference, 2010 Dec 8; India.

[2]. Apache Hadoop http://hadoop.apache.org/

[3]. J. Venner, Pro Hadoop. Apress, June 22, 2009, 440 pages, ISBN: 1430219424

[4]. J. Dean, S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," In Proc. of 6th Symposium on Operating Systems Design and Implementation, San Francisco CA, Dec 04.

[5]. T. White, Hadoop: The Definitive Guide. O'Reilly Media, Yahoo! Press, June 5th, 2009.

[6]. Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST '10)*, Incline Village, Nevada, May 2010.

[7]. P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. "PVFS: A parallel file system for Linux clusters," in Proc. of 4th Annual Linux Showcase and Conference, 2000, pp. 317–327

[8]. W. Tantisiriroj, S. Patil, G. Gibson. "Data-intensive file systems for Internet services: A rose by any other name ..." Technical Report CMUPDL-08-114, Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, PA, October 2008.

[9]. Lustre File System. http://www.lustre.org

[10]. M. K. McKusick, S. Quinlan. "GFS: Evolution on Fast-forward," ACM Queue, vol. 7, no. 7, New York, NY. August 2009.

[11]. FUSE–DFS http://wiki.apache.org/hadoop/MountableHDFS

[12]. S. Vazhkudai, X. Ma, V.W. Freeh, J.W. Strickland, N. Tammineedi, and S. L. Scott, "FreeLoader: Scavenging Desktop Storage Resources for Scientific Data", in Proc. SC, 2005.

[13] MapR http://www.mapr.com/

[14] Cloudera blog http://www.cloudera.com/blog/2009/07/file-appends-in-hdfs/