CS 657 - Paper Review 2

The Hadoop Distributed File System

Nick Alvarez

23 March 2021

1 Clarity

This paper is presented in a concise manner. The architecture of Hadoop is discussed, with information about each of its components, and later moves to the mechanics of I/O operations and replica management. The authors continue with a real-world application at Yahoo! and their findings from this, including statistics on node failure rates in multiple scenarios. Finally, a few performance metrics are examined to better understand its capabilities. There were some issues with clarity when the authors talked about multiple functions within components. They were defined once and sometimes referenced much later in the paper. A graphical representation of, for example, how journaling and checkpoints worked together on the NameNode, or perhaps a chronological list of operations performed on startup and potential error states, would have served as a useful method to convey the ideas.

Presenting one's work in this manner is adequate for detailing system components and their relations to each other, but more real-world examples of the system in use and the types of data it processes would provide some clarity in its intended usage.

2 Problems and Existing Solutions

Accessing large amounts of data, performing operations on it (read/write/append), and ensuring redundant copies of all data can be very costly in an unoptimized file system. When multiple clients can be reading and writing at the same time, the solution needs the flexibility and scalability to handle this. If every operation were to be synchronous, for example, clients would need to wait for all jobs in front of them to finish before theirs could run. Bandwidth, in this sense, would be limited, slowing the average read and write times across the system. For such large amounts of data, often reaching into the petabyte level, this is impractical. Couple this with data backups and/or redundancy, and there will be quite a bottleneck.

Other state of the art solutions were mentioned, including PVFS, Lustre, and GFS. The former two use RAID for data durability, which, while having its advantages, can often be reliability, performance, or space inefficient. The latter two are currently evolving into distributed namespace solutions, set for future releases.

The problems mentioned are important within the realm of "Big Data" and similar fields as efficiency and reliability are paramount when dealing with such large quantities of data. Shvachko et al. use the Web Map as an example such a project requiring this functionality, as it is essential to web searching. Users want a system from which they can access their data quickly and ensure every risk of data loss or corruption is minimized.

These problems are hard due to the very nature of storage systems with tens of thousands of terabytes necessitating unique ways to read and write data, both as a client and for redundancy purposes. Traditional filesystems, such as those on consumer computing equipment, do not expect the user to be working with such large quantities of data, and do not necessarily place their focus on the utmost efficiency and reliability. New filesystems have to be created with a specific purpose to handle the data, and Hadoop is another solution, among others, to the problems.

3 Techniques and New Solutions

Hadoop uses HDFS as its filesystem component. Storing metadata on a dedicated server (the NameNode) and application data on other servers (multiple DataNodes) is the structure of HDFS. The NameNode acts as a directory to all data on the system. Aside from basic metadata like modification time and permissions, the NameNode maintains the mapping of blocks (128 megabyte files) to all of its DataNodes the block has been replicated on. A client requesting data to read or write will go through the NameNode first to find out which DataNodes the block(s) are stored on. DataNodes are simpler, containing only metadata about blocks and the blocks themselves. All data blocks can be replicated (usually three times) across other DataNodes to ensure that if one fails, there are replicas that can be used. Other checks are in place, like heartbeats, sent every three seconds by default, to ensure that a DataNode is still online. If not, replicas of data stored on that DataNode will be made on functional DataNodes. Other checks include namespace identities (to ensure the DataNode is joining the right cluster) and software version (mismatched versions can be incompatible and cause data loss). There are a multitude of other redundancy methods mentioned in the paper, from specifics on where replicas can and cannot be stored, to block scanning for corruption to create new replicas.

Hadoop is better than the state-of-the-art solutions when used for its intended purpose of storing a small number of large files. On a more abstract level, the level of error checking, redundancy methods, and parallel processing sets it apart. Shvachko et al. refers to the HDFS API available to applications like the MapReduce framework, wherein the scheduling of a task to where the

data is located can improve the read performance. Other functionality includes setting a higher replication factor than the default of three. Critical files, for example, may require more replicas to be safe. From top to bottom, Hadoop was designed to make working with large amounts of data simple for users (using the HDFS Client) and maintaining the system easy for administrators (with features like snapshots before upgrades, decommissioning of DataNodes, cluster balancing of storage, and replication management to ensure each block is replicated the correct amount).

The authors go on to demonstrate the performance of Hadoop with three use cases. One, with a contrived benchmark, tests data transfer speeds with random data reads and writes but does not take into account overhead such as task scheduling. A more practical demonstration measures performance in a "busy" environment, where multiple application tasks could be occupying a node. Results were much lower than that of the benchmark test but are a real-world example of Hadoop's performance. They end with performance results from a carefully designed system to achieve the highest performance. Their findings confirm Hadoop's scalability with the number of nodes in a system, as the aggregate I/O increased marginally on a petabyte system versus a terabyte system.

Hadoop, as mentioned previously, was designed to work with a small number of large files. The authors note their assumption of this use case was incorrect and are working to allow multiple NameNodes to share physical storage within the same cluster. They can be isolated to specific sets of applications (some create many small files while others create few large files) in order to mitigate issue where NameNodes cease to function when their memory is near fully used. Furthermore, NameNodes, upon failing, render the whole system unusable (as NameNodes are the directory to all block mappings) and must be restarted. An automated failover to a BackupNode is the author's proposed solution, using Yahoo's Zookeeper technology. Finally, to further increase scalability, they plan to allow greater cooperation between clusters, wherein files could be cached and the replication factor decreased.

4 Suggestions

As discussed earlier, the paper could have benefitted from graphics detailing more of the system architecture (with the inclusion of CheckpointNodes or BackupNodes) or relation of software components. Figure 3, the cluster topology example, did serve as a useful reference for the correlating section, but replica management at a much larger scale was still difficult to imagine. However, for what is still an open-source project Hadoop is a very robust solution for big data management.