www.ijemr.net ISSN (ONLINE): 2250-0758, ISSN (PRINT): 2394-6962

Vandana Publications
IJEMR

Volume-6, Issue-5, September-October 2016
International Journal of Engineering and Management Research
Page Number: 281-286

Hadoop File Management System

Swaraj Pritam Padhy[1], Sashi Bhusan Maharana[2]
[1]Student, Department of Computer Science & Engineering, Centurion University of Technology & Management, Odisha, Parlakhemundi, Gajapati, Odisha, INDIA
[2]Faculty, Department of Computer Science & Engineering, Centurion University of Technology & Management,Odisha, Parlakhemundi, Gajapati, Odisha, INDIA

ABSTRACT

Hadoop is a Java software framework that supports data-intensive distributed applications and is developed under open source license. It enables applications to work with thousands of nodes and peta bytes of data. The two major pieces of Hadoop are HDFS: Hadoop own file system. This is designed to scale to petabytes of storage and runs on top of the file systems of the underlying operating systems.

The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size. We describe the architecture of HDFS and report on experience using HDFS to manage 25 petabytes of enterprise data at Yahoo!. HDFS stores file system metadata and application data separately.

For improved durability, redundant copies of the checkpoint and journal can be made at other servers. During restarts the Name Node restores the namespace by reading the namespace and replaying the journal.

Keywords--- HDFS, Hadoop, Cluster, Hadoop file management system

I. INTRODUCTION

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. A Hadoop frame-worked application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

II. CORELETED STUDY

A. What is Hadoop?

Hadoop is sub-project of Lucene (a collection of industrial-strength search tools), under the umbrella of the Apache Software Foundation. Hadoop parallelizes data processing across many nodes (computers) in a compute cluster, speeding up large computations and hiding I/O latency through increased concurrency. Hadoop is especially well-suited to large data processing tasks (like searching and indexing) because it can leverage its distributed file system to cheaply and reliably replicate chunks of data to nodes in the cluster, making data available locally on the machine that is processing it. Hadoop is written in Java. Hadoop programs can be written using a small API in Java or Python. Hadoop can also run binaries and shell scripts on nodes in the cluster provided that they conform to a particular convention for string input/output. Hadoop provides to the application programmer the abstraction of map and reduce (which may be familiar to those with functional programming experience). Map and reduce are available in many languages, such as Lisp and Python.

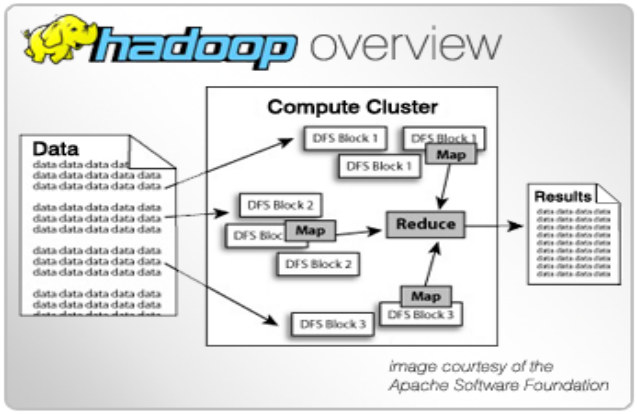

Fig.1 Hadoop over View

III. ABOUT HDFS

A. Hadoop Architecture:

Hadoop framework includes following four modules:

1. Hadoop Common: These are Java libraries and utilities required by other Hadoop modules. These libraries provide file system and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.

2. Hadoop YARN: This is a framework for job scheduling and cluster resource management.

3. Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.

4. Hadoop Map Reduce: This is YARN-based system for parallel processing of large data sets.

We can use following diagram to depict these four components available in Hadoop framework.

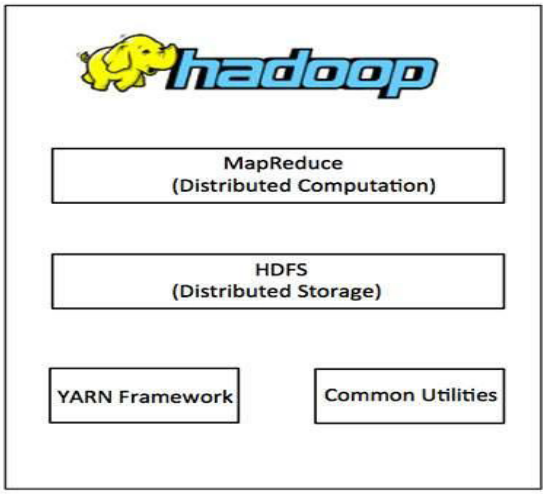

Fig.2 HADOOP Framework

Since 2012, the term "Hadoop" often refers not just to the base modules mentioned above but also to the collection of additional software packages that can be installed on top of or alongside Hadoop, such as Apache Pig, Apache Hive, Apache HBase, Apache Spark etc.

B. Hadoop Applications:

1. Making Hadoop Applications More Widely Accessible.

2. A Graphical Abstraction Layer on Top of Hadoop Applications.

3. With its Eclipse-based graphical workspace, Talend Open Studio for Big Data enables the developer and data scientist to leverage Hadoop loading and processing technologies like HDFS, HBase, Hive, and Pig without having to write Hadoop application code. By simply selecting graphical components from a palette, arranging and configuring them, you can create Hadoop jobs that, for example:

a. Load data into HDFS (Hadoop Distributed File System)

b. Use Hadoop Pig to transform data in HDFS.

c. Load data into a Hadoop Hive based data warehouse

d. Perform ELT (extract, load, transform) aggregations in Hive.

e. Leverage Sqoop to integrate relational databases and Hadoop.

4. Hadoop Applications Seamlessly Integrated.

C. Advantages of Hadoop:

1. Scalable:

Hadoop is a highly scalable storage platform, because it can stores and distribute very large data sets across hundreds of inexpensive servers that operate in parallel. Unlike traditional relational database systems (RDBMS) that can't scale to process large amounts of data, Hadoop enables businesses to run applications on thousands of nodes involving many thousands of terabytes of data.

2. Cost effective:

Hadoop also offers a cost effective storage solution for businesses' exploding data sets. The problem with traditional relational database management systems is that it is extremely cost prohibitive to scale to such a degree in order to process such massive volumes of data. In an effort to reduce costs, many companies in the past would have had to down-sample data and classify it based on certain assumptions as to which data was the most valuable. The raw data would be deleted, as it would be too cost-prohibitive to keep. While this approach may have worked in the short term, this meant that when business priorities changed, the complete raw data set was not available, as it was too expensive to store.

3. Flexible:

Hadoop enables businesses to easily access new data sources and tap into different types of data (both structured and unstructured) to generate value from that data. This means businesses can use Hadoop to derive valuable business insights from data sources such as social media, email conversations. Hadoop can be used for a wide variety of purposes, such as log processing, recommendation systems, data warehousing, and market campaign analysis and fraud detection.

4. Fast:

Hadoop's unique storage method is based on a distributed file system that basically 'maps' data wherever it is located on a cluster. The tools for data processing are often on the same servers where the data is located, resulting in much faster data processing. If you're dealing with large volumes of unstructured data, Hadoop is able to efficiently process terabytes of data in just minutes, and penta bytes in hours.

5. Resilient to failure:

A key advantage of using Hadoop is its fault tolerance. When data is sent to an individual node, that data is also replicated to other nodes in the cluster, which means that in the event of failure, there is another copy available for use.

D. Disadvantages of Hadoop:

As the backbone of so many implementations, Hadoop is almost synonymous with big data.

1. Security Concerns:

Just managing complex applications such as Hadoop can be challenging. A simple example can be seen in the Hadoop security model, which is disabled by default due to sheer complexity. If whoever managing the platform lacks of know how to enable it, your data could be at huge risk. Hadoop is also missing encryption at the storage and network levels, which is a major selling point for government agencies and others that prefer to keep their data under wraps.

2. Vulnerable By Nature:

Speaking of security, the very makeup of Hadoop makes running it a risky proposition. The framework is written almost entirely in Java, one of the most widely used yet controversial programming languages in existence. Java has been heavily exploited by cybercriminals and as a result, implicated in numerous security breaches.

3. Not Fit for Small Data:

While big data is not exclusively made for big businesses, not all big data platforms are suited for small data needs. Unfortunately, Hadoop happens to be one of them. Due to its high capacity design, the Hadoop Distributed File System lacks the ability to efficiently support the random reading of small files. As a result, it is not recommended for organizations with small quantities of data.

4. Potential Stability Issues:

Like all open source software, Hadoop has had its fair share of stability issues. To avoid these issues, organizations are strongly recommended to make sure they are running the latest stable version, or run it under a third-party vendor equipped to handle such problems.

5. General Limitations:

The article introduces Apache Flume, Millwheel, and Google's own Cloud Dataflow as possible solutions. What each of these platforms have in common is the ability to improve the efficiency and reliability of data collection, aggregation, and integration. The main point the article stresses is that companies could be missing out on big benefits by using Hadoop alone.

IV. DISSCUSSION

A. Hadoop Distributed File System:

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS).

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner.

HDFS uses a master/slave architecture where master consists of a single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of DataNodes. The NameNode determines the mapping of blocks to the DataNodes. The DataNodes takes care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by NameNode.

HDFS provides a shell like any other file system and a list of commands are available to interact with the file system. These shell commands will be covered in a separate chapter along with appropriate examples.

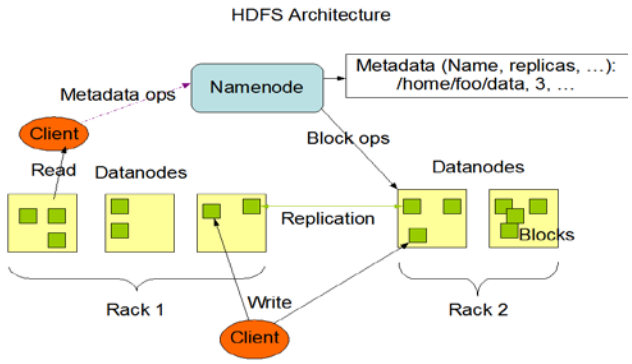

Fig.3 HDFS Architecture

B. Map Reduce:

Hadoop Map Reduce is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

The term Map Reduce actually refers to the following two different tasks that Hadoop programs perform:

1. The Map Task: This is the first task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs).
2. The Reduce Task: This task takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.

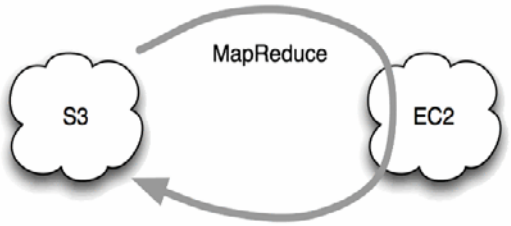

Fig.4 Map Reducing

Typically both the input and the output are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for resource management, tracking resource consumption/availability and scheduling the jobs component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves TaskTracker execute the tasks as directed by the master and provide task-status information to the master periodically.

The JobTracker is a single point of failure for the Hadoop MapReduce service which means if JobTracker goes down, all running jobs are halted.

C. Why is Hadoop important?

1. Ability to store and process huge amounts of any kind of data, quickly: With data volumes and varieties constantly increasing, especially from social media and the Internet of Things (IoT), that's a key consideration.
2. Computing power: Hadoop's distributed computing model processes big data fast. The more computing nodes you use the more processing power you have.
3. Fault tolerance. Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. Multiple copies of all data are stored automatically.
4. Flexibility. Unlike traditional relational databases, you don’t have to preprocess data before storing it. You can store as much data as you want and decide how to use it later. That includes unstructured data like text, images and videos.
5. Low cost. The open-source framework is free and uses commodity hardware to store large quantities of data.
6. Scalability. You can easily grow your system to handle more data simply by adding nodes. Little administration is required.

D. HDFS Client:

User applications access the filesystem using the HDFS client, a library that exports the HDFS filesystem interface.

Like most conventional filesystems, HDFS supports operations to read, write and delete files, and operations to create and delete directories. The user references files and directories by paths in the namespace. The user application does not need to know that filesystem metadata and storage are on different servers, or that blocks have multiple replicas.

When an application reads a file, the HDFS client first asks the NameNode for the list of DataNodes that host replicas of the blocks of the file. The list is sorted by the network topology distance from the client. The client contacts a DataNode directly and requests the transfer of the desired block. When a client writes, it first asks the NameNode to choose DataNodes to host replicas of the first block of the file. The client organizes a pipeline from node-to-node and sends the data. When the first block is filled, the client requests new DataNodes to be chosen to host replicas of the next block. A new pipeline is organized, and the client sends the further bytes of the file. Choice of DataNodes for each block is likely to be different. The interactions among the client, the NameNode and the DataNodes are illustrated.

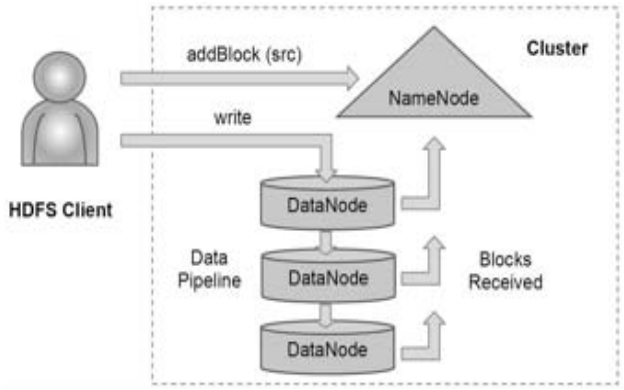

Fig.5 HDFS and HDFS Client

Unlike conventional file systems, HDFS provides an API that exposes the locations of a file blocks. This allows applications like the MapReduce framework to schedule a task to where the data are located, thus improving the read performance. It also allows an application to set the replication factor of a file. By default a file's replication factor is three. For critical files or files which are accessed very often, having a higher replication factor improves tolerance against faults and increases read bandwidth.

E. Features for Sharing HDFS:

As the use of HDFS has grown, the file system itself has had to introduce means to share the resource among a large number of diverse users. The first such feature was a permissions framework closely modeled on the UNIX permissions scheme for file and directories. In this framework, files and directories have separate access permissions for the owner, for other members of the user group associated with the file or directory, and for all other users. The principle differences between Unix (POSIX) and HDFS are that ordinary files in HDFS have neither execute permissions nor sticky bits.

In the earlier version of HDFS, user identity was weak: you were who your host said you are. When accessing HDFS, the application client simply queries the local operating system for user identity and group membership. In the new framework, the application client must present to the name system credentials obtained from a trusted source. Different credential administrations are possible; the initial implementation uses Kerberos. The user application can use the same framework to confirm that the name system also has a trustworthy identity. And

the name system also can demand credentials from each of the data nodes participating in the cluster.

The total space available for data storage is set by the number of data nodes and the storage provisioned for each node. Early experience with HDFS demonstrated a need for some means to enforce the resource allocation policy across user communities. Not only must fairness of sharing be enforced, but when a user application might involve thousands of hosts writing data, protection against applications inadvertently exhausting resources is also important. For HDFS, because the system metadata are always in RAM, the size of the namespace (number of files and directories) is also a finite resource. To manage storage and namespace resources, each directory may be assigned a quota for the total space occupied by files in the sub-tree of the namespace beginning at that directory. A separate quota may also be set for the total number of files and directories in the sub-tree.

While the architecture of HDFS presumes most applications will stream large data

F. Checkpoint Node:

The NameNode in HDFS, in addition to its primary role serving client requests, can alternatively execute either of two other roles, either a Checkpoint Node or a Backup Node. The role is specified at the node startup.

The Checkpoint Node periodically combines the existing checkpoint and journal to create a new checkpoint and an empty journal. The Checkpoint Node usually runs on a different host from the NameNode since it has the same memory requirements as the NameNode. It downloads the current checkpoint and journal files from the NameNode, merges them locally, and returns the new checkpoint back to the NameNode.

Creating periodic checkpoints is one way to protect the file system metadata. The system can start from the most recent checkpoint if all other persistent copies of the namespace image or journal are unavailable. Creating a checkpoint also lets the NameNode truncate the journal when the new checkpoint is uploaded to the NameNode. HDFS clusters run for prolonged periods of time without restarts during which the journal constantly grows. If the journal grows very large, the probability of loss or corruption of the journal file increases. Also, a very large journal extends the time required to restart the NameNode. For a large cluster, it takes an hour to process a week-long journal. Good practice is to create a daily checkpoint.

G. Backup Node:

A recently introduced feature of HDFS is the BackupNode. Like a CheckpointNode, the BackupNode is capable of creating periodic checkpoints, but in addition it maintains an in-memory, up-to-date image of the filesystem namespace that is always synchronized with the state of the NameNode.

The BackupNode accepts the journal stream of namespace transactions from the active NameNode, saves them in journal on its own storage directories, and applies these transactions to its own namespace image in memory. The NameNode treats the BackupNode as a journal store the same way as it treats journal files in its storage directories. If the NameNode fails, the BackupNode's image in memory and the checkpoint on disk is a record of the latest namespace state.

The BackupNode can create a checkpoint without downloading checkpoint and journal files from the active NameNode, since it already has an up-to-date namespace image in its memory. This makes the checkpoint process on the BackupNode more efficient as it only needs to save the namespace into its local storage directories.

The Backup Node can be viewed as a read-only NameNode. It contains all filesystem metadata information except for block locations. It can perform all operations of the regular NameNode that do not involve modification of the namespace or knowledge of block locations. Use of a Backup Node provides the option of running the NameNode without persistent storage, delegating responsibility of persisting the namespace state to the Backup Node.

Cluster:

The cluster administrator can choose to roll back HDFS to the snapshot state when restarting the system. The NameNode recovers the checkpoint saved when the snapshot was created. Data Nodes restore the previously renamed directories and initiate a background process to delete block replicas created after the snapshot was made. Having chosen to roll back, there is no provision to roll forward. The cluster administrator can recover the storage occupied by the snapshot by commanding the system to abandon the snapshot, thus finalizing the software upgrade.

How Does Hadoop Work?

Stage 1:

A user/application can submit a job to the Hadoop (a hadoop job client) for required process by specifying the following items:

1. The location of the input and output files in the distributed file system.
2. The java classes in the form of jar file containing the implementation of map and reduce functions.
3. The job configuration by setting different parameters specific to the job.

Stage 2:

The Hadoop job client then submits the job (jar/executable etc) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

Stage 3:

The TaskTrackers on different nodes execute the task as per MapReduce implementation and output of the reduce function is stored into the output files on the file system.

Data Lake:

Data lakes support storing data in its original or exact format. The goal is to offer a raw or unrefined view of data to data scientists and analysts for discovery and analytics. It helps them ask new or difficult questions without constraints. Data lakes are not a replacement for data warehouses. In fact, how to secure and govern data lakes is a huge topic for IT. They may rely on data federation techniques to create a logical data structures.

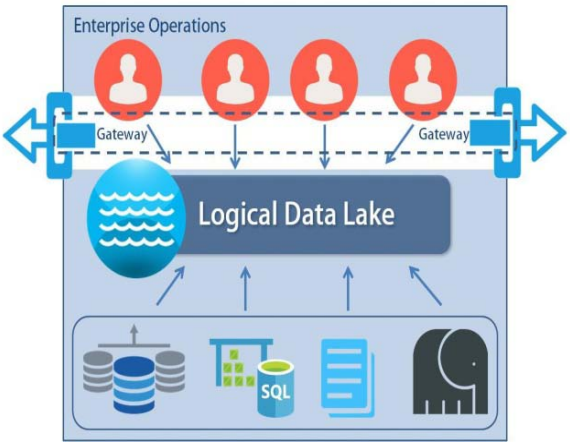

Fig.6 Logical Data Lake

V. CONCLUSION

This section presents some of the future work that the Hadoop team at Yahoo is considering; Hadoop being an open source project implies that new features and changes are decided by the Hadoop development community at large. The Hadoop cluster is effectively unavailable when its NameNode is down. Given that Hadoop is used primarily as a batch system, restarting the NameNode has been a satisfactory recovery means. However, we have taken steps towards automated failover. Currently a BackupNode receives all transactions from the primary NameNode. This will allow a failover to a warm or even a hot BackupNode if we send block reports to both the primary NameNode and BackupNode. A few Hadoop users outside Yahoo! have experimented with manual failover.

Our plan is to use Zookeeper, Yahoo's distributed consensus technology to build an automated failover solution. Scalability of the NameNode has been a key struggle. Because the NameNode keeps all the namespace and blocklocations in memory, the size of the NameNode heap has limited the number of files and also the number of blocks addressable. The main challenge with the NameNode has been that when its memory usage is close to the maximum theNameNode becomes unresponsive due to Java garbage collection and sometimes requires a restart.

REFERENCE

[1] www.wikipedia.org
[2] www.studymafia.org
[3] Apache Hadoop. http://hadoop.apache.org/
[4] 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.