Cassandra File System Over Hadoop Distributed File System

Mr. Ashish A. Mutha
ME, CSE,
PRMIT&R College,
Amravati, India
ashishmutha10@gmail.com

Miss. Vaishali M. Deshmukh
Assistant Professor,
PRMIT&R College,
Amravati, India
msvmdeshmukh@rediff.com

Abstract**—**Cassandra is an open source distributed database management system is designed to handle large amounts of data across many commodity servers, provides a high availability with no single point of failure. Cassandra will be offering the robust support for clusters spanning multiple data centers with asynchronous masterless replica which allow low latency operations for all the clients. NoSQL data stores target the unstructured data, which nature has dynamic and a key focus area for "Big Data" research. New generation data can prove costly and also unpractical to administer with databases SQL, due to lack of structure, high scalability and needs for the elasticity. NoSQL data stores such as MongoDB and Cassandra provide a desirable platform for fast and efficient for data queries. The Hadoop Distributed File System is one of many different components and projects contained within the community Hadoop ecosystem. The Apache Hadoop project defines Hadoop-DFS as “the primary storage system which is used by Hadoop applications” that enables “reliable, extremely rapid computations”. This paper was providing high-level overview of how Hadoop-styled analytics (MapReduce, Pig, Mahout and Hive) can be run on data contained in Apache Cassandra without the need for Hadoop-DFS.

Keywords: *Cassandra: CFS; Benefits of CFS; overview of HDFS.*

*****

I. Introduction

The Cassandra File System (CFS) is a Hadoop-DFS compatible file system built to replace the traditional Hadoop NameNode and Secondary, NameNode and DataNode daemons [1]. The main aim of design goals for the Cassandra File System were to first, it simplify the operational overhead of Hadoop by removing the single points of failure in the Hadoop NameNode. And second is to offer easy Hadoop integration for Cassandra users (one distributed system is enough for that). The Apache Hadoop project defines Hadoop-DFS as: “the primary storage system which used by Hadoop applications. Hadoop-DFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable, reliable, extremely rapid computations” to it. Hadoop utilizes a scale-out architecture that makes the use of commodity servers has to configure as a cluster, and where each of these servers possesses inexpensive internal disk drives. As per Apache site states that data in Hadoop is broken down into blocks and spread throughout a cluster. Once that was happen, MapReduce tasks can be carried out on the smaller subsets of the data that may make up a very large dataset overall, thus to accomplish the type of scalability needed for big data processing. In general, this divide-and-conquer strategy of processing data is nothing really new, but the combination of Hadoop-DFS being open source software (which overcomes the need for the high-priced specialized storage solutions) and its ability to carry out some degree of automatic failover and redundancy make it so popular for the modern businesses looking for data warehouse batch analytics solutions. That’s why? This is just one reason why the Hadoop market is expected to grow at an eye-popping compound annual growth rate (CAGR) of 58 percent until 2018. However, what these businesses are most interested in not Hadoop underlying storage structure, but rather than what it facilitates in delivering: a cost-effective means for analyzing and processing vast amounts of data for data warehouse use cases. But what about analytics needed for line-of-business application data? This paper provides a high-level overview of how DataStax uses Apache Cassandra to run analytics on Cassandra data that comes from line-of-business applications.

II. Overview Of HDFS

A typical Hadoop deployment with Hadoop Distributed File System resembles the following:

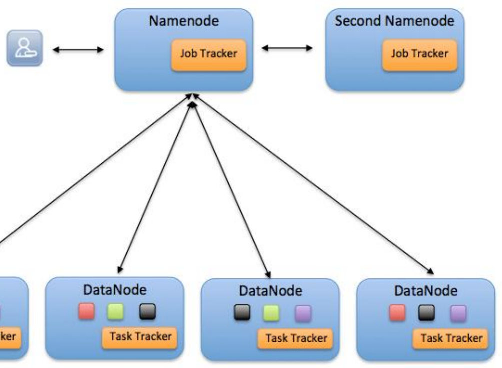

Figure 1: HDFS with Standard Hadoop Deployment

Hadoop and HDFS utilize master-slave architecture. HDFS is written in Java, with an HDFS cluster consisting of a primary NameNode – a master server that manages the file system namespace and also regulates access to data by clients. The optional secondary Name Node for failover purposes also may be configured. In addition, there are a number of DataNodes. And it consist a one-to-one relationship between a DataNode and physical machine. Each of this DataNode manages the storage attached to the boxes that it runs on. Hadoop Distributed File System uses a file system namespace that enables data to be stored in files. Each file is divided into one or more blocks and which are then divvied up across a set of DataNodes. The NameNode is responsible for tasks such as opening, renaming, data directories and closing files. It also tackles the job of mapping blocks to DataNodes, which is then responsible for managing incoming Input/Output requests from clients. DataNode handle the block replication, creation, and removal of data when instructed by the NameNode.

Benefits of HDFS

There is little debate that HDFS provides a number of benefits for those who choose to use it. Below are some of the most commonly cited? [8, 10]

Built-In Redundancy and Failover: Hadoop distributed file system supplies out-of-the-box redundancy and failover capabilities that require little to no manual intervention i.e. depending on the use case. Having such features built into the storage layer allows system administrators and developers to concentrate on other responsibilities versus having to create monitoring systems or/and programming routines to compensate for another set of storage software that lacks those capabilities. Moreover, with downtime being a real threat to many modern businesses' bottom line, features that minimize outages and contribute to keeping a batch analytic data store up, operational, and feeding any online system that requires its input are welcomed by all professionals.

Big Data Capable: The hallmark of HDFS is its ability to tackle big data use cases and most of the characteristics that comprise them (data velocity, volume and their variety). The rate at which HDFS can supply data to the programming layers of Hadoop equates to faster batch processing times and quicker answers to complex analytic questions.

Portability: Any tenured data professional can relay horror stories of having to transfer, migrate, and convert huge data volumes between disparate storage/software vendors. One benefit of HDFS is its portability between various Hadoop distributions which helps to minimize vendor lock-in.

Cost-Effective: As previously stated that, Hadoop-DFS is one of the open source software, the system which translates into real cost savings for its users. As many companies can attest, high-priced storage solutions can take a significant bite out of IT budgets and are many times completely out of reach for small or startup companies. Other benefits of HDFS exist, but the four above are the primary reasons why many users deploy HDFS as their analytic storage solution.

III. CASSANDRA

Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers and also providing a high availability with no single point of failure in Cassandra. Cassandra also offers a robust support for clusters spanning multiple data centers [1, 15] with asynchronous masterless replication which allowing low latency operations for all clients. Apache Cassandra is a massively scalable NoSQL database and Used today by numerous modern businesses to manage their critical data infrastructure. Cassandra is also known for the solution to technical professionals turn to when they need a real-time NoSQL database that supplies high performance at massive scale and which is never goes down [11].

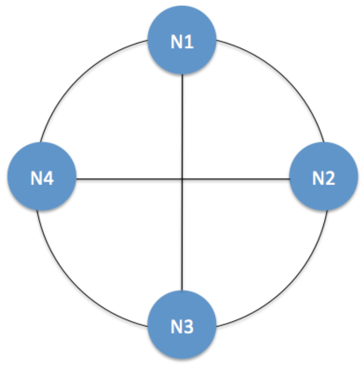

Fig 2: Simple Cassandra Network

Rather than using a legacy master-slave or a manual and difficult-to-maintain sharded design, Cassandra is a peer-to-peer distributed "ring" architecture that is much more elegant and easy to setup and maintain. In Cassandra, all nodes are the same and there is no concept of a master node with all nodes communicating with each other via a gossip protocol. Cassandra's built-for-scale architecture means that is capable of handling petabytes of information and thousands of concurrent users/operations per second across one to many data centers as easily as it can manage much smaller amounts of data and user traffic. It means that also unlike other master-slave or sharded systems, Cassandra system has no single point of failure system and therefore is capable of offering true continuous availability. It is not a file system at all but it is an open source and it store NoSQL key-value [2]. Cassandra has become a viable alternative to HDFS for web applications that rely on fast data access. DataStax is a startup commercializing of the Cassandra database has fused Hadoop atop Cassandra to provide web applications fast access to data processed by Hadoop. Hadoop have a fast access to data streaming into Cassandra from web users.

IV. CASSANDRA FILE SYSTEM

What is the Cassandra File System (CFS)?

The Cassandra File System (CFS) was designed by DataStax Corporation to easily run analytics on Cassandra data. Now it is implemented as part of DataStax Enterprise, which combines Apache Cassandra and Solr together into a unified big data platform, CFS provides the storage foundation that makes running Hadoop-styled analytics on Cassandra data hassle-free. The main design goals for the Cassandra File System were to first, simplify operational overhead of Hadoop by removing the single points of failure in the Hadoop NameNode. And second, to offer easy Hadoop integration for Cassandra users (one distributed system is enough) [5].

How Does CFS Work?

In contrast to master-slave architecture like HDFS, CFS is based on Cassandra, so the implementation is peer-to-peer and "masterless". User is able to create a cluster that seamlessly stores real-time data in Cassandra; it performs analytic operations on that same data and also handles enterprise search operations. Cassandra's built-in replication transparently takes care of replicating the data among all real time, analytic and search nodes. A user may configure any type of cluster they desire.

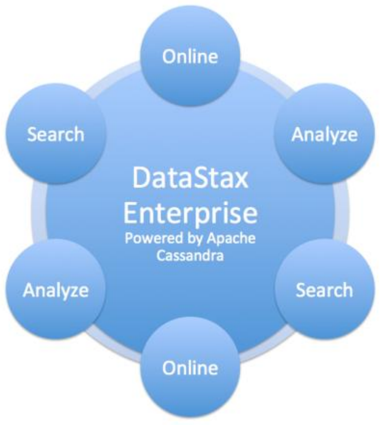

Figure 3: A Simple DataStax Cluster

[6] CFS stores metadata information regarding analytics data in a Cassandra keyspace, which is analogous to a database in the relational database management system (RDBMS) world. Two Cassandra column families (like tables in an RDBMS) in the keyspace contain the actual data. The data contained in these column families is replicated across the cluster to ensure data protection and fault tolerance. The column families mirror the two primary HDFS services. The inode column family replaces the HDFS NameNode service, which tracks each data file's metadata and block locations on the participating of the analytics nodes. And captured information in this column family includes filename, parent path, group, user, permissions, file type and a list of block ids that make up the file. For block ids, it uses Time UUID, so blocks are ordered sequentially in a natural way and its makes supporting HDFS functions like append() easy. The sblocks column family supplants the HDFS DataNode service that stores file blocks. This column family stores the actual contents of any file that is added to an analytics node. Each row in sblocks represents a block of data associated with a row in the inode column family. Each row key is a block Time UUID from an inode row. The columns are the time ordered compressed sub-blocks that, when it is decompressed and combined, equal to one Hadoop-DFS block.

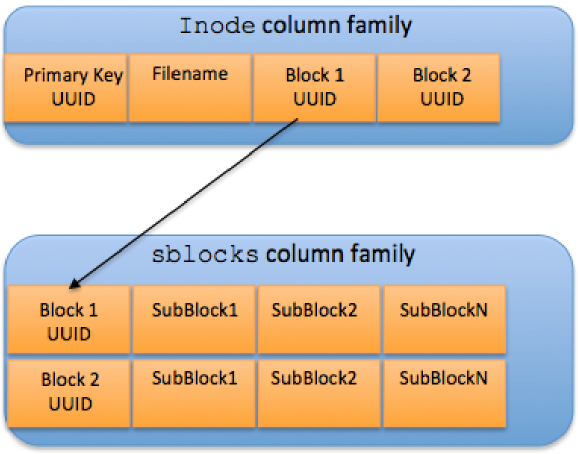

Figure 4: CFS Column Families

When data is added to an analytics node, Cassandra File System writes the static metadata attributes to the inode column family. Then it allocates a new sblocks row object, reads a chunk of that data (is controlled via the Hadoop parameter fs.local. block.size), which splits it into sub-blocks (is controlled via the parameter cfs.local.subblock.size), and compresses them via Google's snappy compression. Once a specific block is complete then its block id is written to the inode column family row and the sub-blocks are written to Cassandra with the block id as the row key and the sub-block ids as the columns [7]. Reads are handled in a straightforward manner. When query request comes into an analytics node, CFS reads the inode information and finds the block and sub-block(s) needed to satisfy the request.

V. Benefits Of CASSANDRA FILE SYSTEM

CFS is built on a proven and trusted technology (Apache Cassandra) that powers many applications all over the globe and world, and possesses a reputation known for scaling and performing extremely well under challenging workloads [3]. Next, it should be understood that CFS is completely transparent to any developer or end user. There are no changes to any Hive, MapReduce, Pig, Mahout or any other routines that run against CFS. [4]There are, however, a number of benefits derived from using CFS over HDFS.

A. Simpler Deployment

With CFS, there is no need for any master-slave failover configurations, no zookeeper requirements, and no complex storage requirements for storage area networks (SANs). Instead, a cluster can be set up and installed in a few minutes, with all CFS configurations being handled automatically when a inside of DataStax Enterprise marked for analytics is started for the first time. By using CFS, in essence, three traditional Hadoop services (NameNode, Secondary NameNode, and DataNode) are replaced with one easy-to-understand and use fault tolerant component.

B. Better Availability

Continuous availability for analytics in a database cluster in CFS is maintained without the need for any shared storage solution (e.g. SANs). Instead, a cluster can consist of vanilla, white-box hardware with local storage and still meet any high-availability SLA requirement of it. Cassandra's redundancy and replication provides complete customization with respect to how many copies of data should be maintained in a cluster, thus ensuring constant uptime and no chance for data loss.

C. Multi-Data Center Support

Many modern businesses need to run analytic operations that span more than one data center. CFS's continuous availability benefits include supporting multi-data center, cloud, and hybrid (on-premise and cloud) environments. CFS supports running a single database cluster across as many data centers as desired, with any node in the cluster being able to service reads and writes. Moreover, an architect can create multiple CFS keyspaces so that each data center has its own local copy of all the data it needs. A Job Tracker for each data center can also be configured so each location has its own for handling MapReduce and other analytic processing jobs

D. No Shared Storage Requirment For Failover

The Hadoop documentation is clear on the need for a shared storage solution to support failover: Shared storage – you will need to have a shared directory which has NameNode machines can have read/write access to both. Currently only a single shared edits directory is supported. Typically, this is a remote filer which supports NFS and is mounted on each of the NameNode machines. Thus, availability of the system is limited by the availability of this shared edits directory and therefore, in order to remove all single points of failure there needs to be redundancy for the shared edits directory. Specifically, the multiple network paths of storage and redundancy in the storage itself (disk, network and power). Because of this; it is recommended that the shared storage server to be a high-quality dedicated NAS appliance rather than a simple Linux server. No such requirement is needed for CFS failover as everything is automatically and transparently handled by Cassandra.

E. Full Data Integration

CFS provides the ability to have one big data platform that handles real-time, analytic, and enterprise search workloads in one cluster without any workload affecting the other where data or compute resources are to be concerned. Instead, the full mixed workload support is built and transparently handled by DataStax Enterprise. This benefit results in the elimination of data “silos” in organizations and the need to create and maintain costly ETL routines to move data between different silos. Any data written to Cassandra is replicated to analytics and search nodes, and vice versa. Further, even output results from analytics tasks may be replicated. For example, a Hive query on analytic nodes that takes some time to complete and produces a result set is able to have that result set replicated over to Cassandra nodes for real-time query access.

F. MapReduce Support

Cassandra has Hadoop integration with MapReduce support. Also support to the Apache Hive and Apache Pig.

G. Tunable Consistency

Writes and reads offer a tunable level of consistency, from all the way "writes never fail" to "all replicas to be readable for block", with the quorum level in middle.

H. Commodity Hardware Support

CFS runs well on commodity hardware and requires no special server or network equipment to be purchased.

VI. Conclusion

While HDFS is a good solution for providing cost-effective storage for Hadoop implementations devoted to data warehouse systems, Cassandra file system delivers the ability to run analytics on Cassandra data that comes from line-of-business applications. This world has researched and developed a scalable high-performance storage system for user inventory items Cassandra. Cassandra is the most popular wide column store. CFS is built on provender and trusted technology that powers many applications all over the world, and possesses a reputation known for scaling and performing extremely well under workloads and it should be understood that CFS is completely transparent to any developer or end user. There are no changes to any MapReduce, Hive, Pig, Mahout, or other routines run rather than CFS. The way Cassandra manages the persistent state in the face (or case) of these failures drives; the reliability and scalability of the software systems relying on this service. While in many ways Cassandra resembles a database and shares many design and implementation strategies therewith, it does not support a full relational data model instead of that it provides clients with a simple data model that supports dynamic control over data lay- out and format. Cassandra file system was designed to run on cheap commodity hardware and will handle high write through- put while not sacrificing read efficiency.

.

Acknowledgment

This work was guided by professor Miss. Vaishali Deshmukh. I would like to thank her for their gaudiness and help.

References

[1] " Apache Cassandra " http://en.wikipedia.org/wiki/Apache_Cassandra.2014.

[2] “Designing performance monitoring tool for NoSQL Cassandra distributed database”. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6360579&queryText%3DCASSANDRA, 2012 IEEE.

[3] “Cassandra: flexible trust management, applied to electronic health records”, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=1310738&queryText%3DCASSANDRA, IEEE.

[4] "The NoSQL Principles and Basic Application of Cassandra Model", http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6394574&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6394574,2012 IEEE.

[5] http://www.datastax.com/dev/blog/cassandra-file-system-design.

[6] http://www.datastax.com/docs/0.8/ddl/index#data-model.

[7] http://www.datastax.com/dev/blog/cassandra-file-system-design.

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

[9] “A. Lakshman and P. Malik. Cassandra: structured storage system on ap2p network. In Proceedings of the 28th ACM symposium on Principles of distributed computing, PODC ’09, pages 5–5, New York, NY, USA, 2009. ACM.

[10] “An Evaluation of Cassandra for Hadoop”, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6676732, 2013 IEEE.

[11] http://nosqlsummer.org/paper/cassandra.

[12] http://tdwi.org/articles/2013/08/20/datastax-hadoop-cassandra.aspx.

[13] http://www.slideshare.net/planetcassandra/cfs-cassandra-backed-storage-for-hadoop-20041145.

[14] http://blog.octo.com/en/introduction-to-datastax-brisk-an-hadoop-and-cassandra-distribution/.

[15] https://wiki.apache.org/cassandra/ArchitectureInternals.