Big Data and Hadoop

Bhavna Gahlot[1], ,
[1]M.Tech
Department of Computer Science and Engineering
Gitam, Mdu Rohtak

Mr. Ashish Kumar Sharma[2]
[2]Assistant Professor
Department of Computer Science and Engineering and C.F.I.S,
Gitam, Mdu

Mr. Jony Birla[3]
[2]Assistant Professor
Department of Computer Science and Engineering,
Gitam, Mdu Rohtak

Big data is an evolving term that describes any voluminous amount of structured, semi structured and unstructured data that has the potential to be mined for information. Big data can be structured, unstructured or semi-structured, resulting in incapability of conventional data management methods. Data is generated from various different sources and can arrive in the system at various rates. In order to process these large amounts of data in an inexpensive and efficient way, parallelism is used. Big Data is a data whose scale, diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it. Hadoop is the core platform for structuring Big Data, and solves the problem of making it useful for analytics purposes. Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance.

Keywords - Big Data, Hadoop, Map Reduce, HDFS, Hadoop Components

I INTRODUCTION

A. Big Data:

Big data is a term that refers to data sets or combinations of data sets whose size, complexity, and rate of growth (velocity) make them difficult to be captured, managed, processed or analyzed by conventional technologies and tools, like as relational databases and desktop statistics or visualization packages, within the time necessary to make them useful. While the size used to determine whether a particular data set is considered big data is not firmly defined and continues to change over time, most analysts and practitioners currently refer to data sets from '30-50' terabytes(10 12 or 1000 gigabytes per terabyte) to multiple petabytes (1015 or 1000 terabytes per petabyte). It can be decomposed into 3 layers, including Infrastructure Layer, Computing Layer, and Application Layer from top to bottom[1].

B. 3 Vs of Big Data

Volume of data: "Volume refers to amount of data". Volume of data stored in enterprise repositories have grown from megabytes and gigabytes to petabytes.

Variety of data: "Different types of data and sources of data". Data variety exploded from structured and legacy data stored in enterprise repositories to unstructured, semi structured, audio, video, XML etc.

Velocity of data:"Velocity refers to the speed of data processing". For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.

C. Problem with Big Data Processing

i). Heterogeneity and Incompleteness When humans consume information, a great deal of heterogeneity is comfortably tolerated. However, machine analysis algorithms expect homogeneous data, and cannot understand nuance. In consequence, data must be carefully structured as a first step in (or prior to) data analysis. Computer systems work most efficiently if they can store multiple items that are all identical in size and structure. Efficient representation, access, and analysis of semi-structured data require further work.

ii). Scale Of course, the first thing anyone thinks of with Big Data is its size. After all, the word "big" is there in the very name. Managing large and rapidly increasing volumes of data has been a challenging issue for many decades. In the past, this challenge was mitigated by processors getting faster, following Moore's law, to provide us with the resources needed to cope with increasing volumes of data. But, there is a fundamental shift underway now: data volume is scaling faster than compute resources, and CPU speeds are static.

iii). Timeliness The flip side of size is speed. The larger the data set to be processed, the longer it will take to analyze. The design of a system that effectively deals with size is likely also to result in a system that can process a given size of data set faster. However, it is not just this speed that is usually meant when one speaks of Velocity in the context of Big Data. Rather, there is an acquisition rate challenge

iv). Privacy The privacy of data is another huge concern, and one that increases in the context of Big Data. For electronic health records, there are strict laws governing what can and cannot be done. For other data, regulations, particularly in the US, are less forceful. However, there is great public fear regarding the inappropriate use of personal data, particularly through linking of data from multiple sources. Managing privacy is effectively both a technical and a sociological

problem, which must be addressed jointly from both perspectives to realize the promise of big data.

v). Human Collaboration In spite of the tremendous advances made in computational analysis, there remain many patterns that humans can easily detect but computer algorithms have a hard time finding. Ideally, analytics for Big Data will not be all computational rather it will be designed explicitly to have a human in the loop. The new sub-field of visual analytics is attempting to do this, at least with respect to the modeling and analysis phase in the pipeline. In today's complex world, it often takes multiple experts from different domains to really understand what is going on. A Big Data analysis system must support input from multiple human experts, and shared exploration of results. These multiple experts may be separated in space and time when it is too expensive to assemble an entire team together in one room. The data system has to accept this distributed expert input, and support their collaboration.

II HADOOP: SOLUTION FOR BIG DATA PROCESSING

"Hadoop" is a Programming framework used to support the processing of large data sets in a distributed computing environment. Hadoop was developed by Google's MapReduce that is a software framework where an application break down into various parts. The Current ApacheHadoop ecosystem consists of the Hadoop Kernel, Map Reduce, HDFS and numbers of various components like Apache Hive, Base and Zookeeper. HDFS and Map Reduce are explained in following points.

III HDFS ARCHITECTURE:

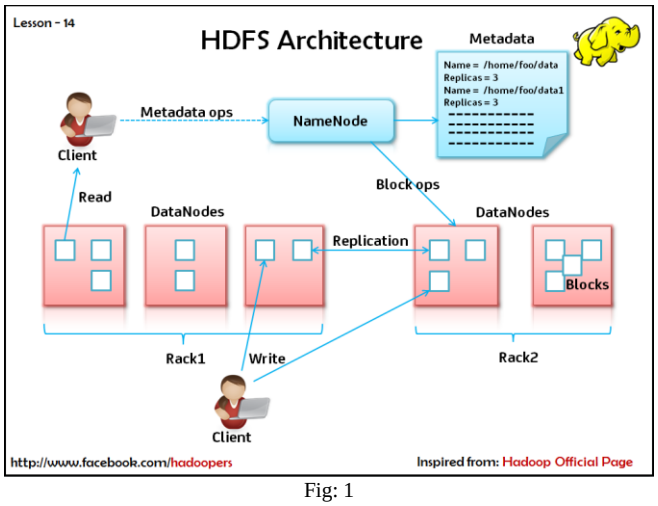

Fig: 1

The placement of the replicas is paramount to HDFS reliability and performance. Having the opportunity to optimize the replica placement scenarios is considered a feature that distinguishes HDFS from most of the other distributed file systems. It has to be pointed out though that the process of optimizing the replica placements requires tuning and experience. A rack-aware replica placement policy focuses on improving data reliability, availability, as well as to optimize network bandwidth utilization. Most HDFS installations execute in a cluster environment that encompasses many racks. Inter-rack node communication travels through switches. In most configurations, the bandwidth potential among nodes in the same rack is greater than the network bandwidth among nodes hosted in different racks. It has to be pointed out that the rack ID for the DataNodes can be manually defined and maintained. One policy option would be to place replicas on individual racks, preventing data loss scenarios when an entire rack fails, while allowing for aggregate bandwidth usage from multiple racks while reading data. Such a policy evenly distributes the replicas in the cluster, nicely focusing on load balancing and component failure scenarios. However, this policy increases the cost of write operations, as each write request requires transferring blocks to multiple racks[2].

For a common case that operates on a replication factor of 3, the HDFS placement policy is to store 1 replica on a node in the local rack, another replica on a different node in the same local rack, and the last replica on a different node in a different rack. This policy addresses the inter-rack write traffic scenario discussed above and generally improves the write performance. The chance of a rack failure is far less than a node failure and hence, this policy does not impact data reliability and availability guarantees. However, it does reduce the aggregate network bandwidth used when reading data, as a block is placed in only two unique racks rather than three. With this policy, the replicas of a file do not evenly distribute across the racks. 2/3 of the replicas are on one rack, and the other 3rdis evenly distributed across the remaining racks. This policy improves write performance without compromising data reliability or read performance. To minimize network bandwidth consumption and maximize read latency, HDFS' design objective is to process any read request from the data replica closest to the read task. Hence, if there is a replica on the same rack as the read task, that replica represents the preferred data location for the read request. In large HDFS environments that may span multiple data centers, the goal is to operate on a replica that is available in the local data center where the read task is issued.

IV HADOOP MAP REDUCE ARCHITECTURE:

The Hadoop Map Reduce MRv1 framework is based on a centralized master/slave architecture. The architecture utilizes a single master server (Job Tracker) and several slave servers (Task Tracker's). Please see Appendix A for a discussion on the Map Reduce MRv2 framework. The Job Tracker represents a centralized program that keeps track of the slave nodes, and provides an interface infrastructure for job submission. The Task Tracker executes on each of the slave nodes where the actual data is normally stored. In other words, the Job Tracker reflects the interaction point among the users and the Hadoop framework. Users submit Map Reduce jobs to the Job Tracker, which inserts the jobs into the pending jobs queue and executes them (normally) on a FIFO basis. The Job Tracker manages the map and reduce task assignments with the Task Tracker's. The Task Tracker's execute the jobs based on the instructions from the Job Tracker and handle the data movement between the map and reduce phases, respectively. Any map/reduce construct basically reflects a special form of a Directed Acyclic Graph (DAG). A DAG can execute anywhere in parallel, as long as one entity is not an ancestor of another entity. In other words,

parallelism is achieved when there are no hidden dependencies among shared states. In the MapReduce model, the internal organization is based on the map function that transforms a piece of data into entities of [key, value] pairs. Each of these elements is sorted (via their key) and ultimately reaches the same cluster node where a reduce function is used to merge the values (with the same key) into a single result (see code below).

V HDFS BLOCK REPLICATION

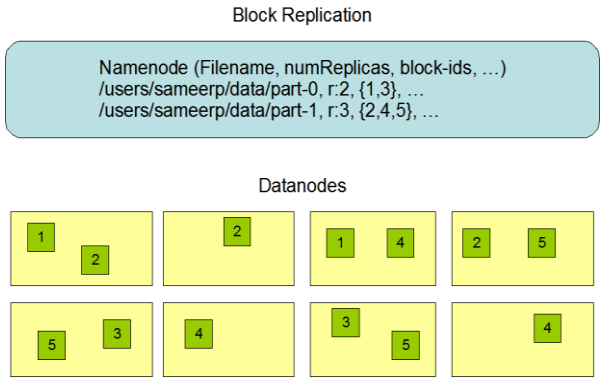

Fig: 2

At startup, the NameNode enters a special (transient) state that is labeled the safe mode. While in this state, the NameNode accepts heartbeat and block report messages from the DataNodes. As already discussed the block report lists the data blocks that the DataNodes are hosting. Each data block is linked to a specific (minimum) number of replicas. A data block is considered fully replicated when the minimum number of replicas for that particular data block has been confirmed with the NameNode. After a certain percentage (this is tunable) of fully replicated data blocks is reached, the NameNode stalls for another 30 seconds, and afterwards exits the safemode state. The next task for the NameNode is to compile a list of data blocks (if any) that still have fewer than the specified number of replicas available, and to replicate these data blocks unto some other DataNodes. The NameNode retains a transaction log to persistently record every change made to file system metadata. To illustrate, creating a new file or changing the replication factor for a file triggers a record insert operation into the editlog. The edit log file itself is stored in the NameNode's local host file system. The HDFS namespace is maintained by the NameNode, as the entire namespace, including the data block file mapping and file system properties are stored in the HDFS fsimage file. Similar to the editlog file, the fsimage file is located in the NameNode's local file system. The NameNode retains an image of the file system namespace and the block map file in memory. As the NameNode boots up, the fsimage and editlog files are read from disk and all the (potential) editlog transactions are mapped into the fsimage in-memory copy and then flushed out to a new version of the fsimage file. At that point, the old editlog file is truncated, as the transactions have been securely applied to the persistent fsimage file (this process is labeled as a checkpoint). The fsimage and the editlog files are considered core HDFS data structures. A corruption of these files can cause the HDFS instance to become non-functional. Hence, a NameNode should be configured so that multiple copies of the two files are being maintained. In other words, any update to either the fsimage or editlog triggers a synchronous update to multiple copies of the 2 files[3]. As most Hadoop/HDFS applications are not metadata intensive, maintaining several copies of the 2 files is not considered a significant performance issue. The HDFS communication protocols are stacked on top of TCP/IP. A client system basically establishes a TCP connection (via a configurable port) to the NameNode, utilizing the client protocol as the communication entity. The DataNodes communicate with the NameNode via the DataNode protocol. A remote procedure call (RPC) wrapper encompasses the client, as well as the DataNode protocols, respectively. By design, the NameNode never initiates any RPC requests. Instead, the NameNode only responds to RPC requests that are issued by either the DataNodes or the clients[4].

It has to be pointed out that a client request to create a file does not reach the NameNode instantaneously, as the HDFS client caches the file data in a temporary local file. Any application write requests are transparently redirected to this temporary local file until the size of the local file reaches the HDFS block size. At that point, the client contacts the NameNode which adds the file name into the file system hierarchy and allocates an HDFS data block. Next, the NameNode responds to the client request by identifying a DataNode and a target data block. After that, the client flushes the data block from the temporary local file to the specified DataNode. When a file is closed, any (potentially) un-flushed data being held in the temporary local file is transferred to the DataNode as well. After the “close()” operation, the client informs the NameNode that the file has been closed. That indicates to the NameNode to commit the file creation operation into persistent storage. This client-side caching architecture was chosen to minimize network congestion and to optimize network throughput. To illustrate, based on the discussed write example, the assumption made is that the HDFS file is configured with a replication factor of 3. As the local file accumulated enough data for an entire block, the client retrieves a list from the NameNode that outlines the DataNodes that will host a replica block. Based on the list, the client starts transferring data to the first DataNode. The first DataNode receives the data in small chunks (normally 4KB blocks), writes each portion of the data block to a local repository, and then transfers that portion of the data to the second DataNode on the list. The second DataNode operates in the same way, and hence transfers its data chunks to the 3d DataNode on the list. Ultimately, the 3d DataNode receives the data and writes the data chunks into its local repository. In other words, depending on the order, a DataNode may simultaneously be receiving data from a previous node while sending data to the next DataNode in the pipeline.

VI THE HADOOP SCHEDULERS

Since the pluggable scheduler framework (similar to the Linux IO schedulers) was introduced, several different scheduler algorithms have been designed, developed, and made available to the Hadoop community. In the next few paragraphs, the FIFO, the Fair, as well as the Capacity schedulers are briefly introduced.

 The FIFO Scheduler -> FIFO reflects the original Hadoop scheduling algorithm that was integrated into the JobTracker framework. With FIFO scheduling, a JobTracker basically just pulls the oldest job from the work queue. The FIFO scheduling approach has no concept of either job priority or job size, but is rather simple to implement and efficient to execute (very low overhead).

 The Fair Scheduler -> Originally, the Fair scheduler was developed by Facebook. The fundamental design objective for the Fair scheduler revolves around the idea to assign resources to jobs in a way that (on average) over time, each job receives an equal share of the available resources. With the Fair scheduler, there is a certain degree of interactivity among Hadoop jobs, permitting a Hadoop cluster to better response to the variety of job types that are submitted over time. From an implementation perspective, a set of pools is setup, and the jobs are placed into these pools and so are made available for selection by the scheduler. Each pool operates on shares to balance the resource usage among the jobs in the pools. The heuristic used is that the more shares the greater the resource usage potential to execute the jobs. By default, all pools are setup with equal shares, but configuration based pool share adjustments can be made based on job types. The number of concurrent active jobs can be constrained to minimize congestion and to allow the workload to be processed in a timely manner. To ensure fairness, each user is assigned to a pool. Regardless of the shares that are assigned to the pools, if the system is underutilized (based on the current workload), the active jobs receive the unused shares (the shares are split among the current jobs). For each job, the scheduler keeps track of the compute time. Periodically, the scheduler examines the jobs to calculate the delta between the actual compute time received and the compute time that the job should have received. The results reflect the deficit matrix for the tasks. It is the scheduler's responsibility to schedule the task with the greatest deficit.

 The Capacity Scheduler -> Originally, the Capacity scheduler was developed by Yahoo. The design focus for the Capacity scheduler was on large cluster environments that execute many independent applications. Hence, the Capacity scheduler provides the ability to provide a minimum capacity guarantee, as well as to share excess capacity among the users. The Capacity scheduler operates on queues. Each queue can be configured with a certain number of map and reduce slots. Further, each queue can be assigned a guaranteed capacity while the overall capacity of the cluster equals to the sum of all the individual queue capacity values. All the queues are actively monitored and in scenarios where a queue is not consuming its allocated capacity potential, the excess capacity can be temporarily allocated to other queues. Compared to the Fair scheduler, the Capacity scheduler controls all the prioritizing tasks within a queue. In general, higher priority jobs are allowed access to the cluster resources earlier than lower priority jobs. With the Capacity scheduler, queue properties can be adjusted on-the-fly, and hence do not require any disruptions in the cluster usage/processing [1].

While not considered a scheduler per se, Hadoop also supports the scheme of provisioning virtual cluster environments (within physical clusters). This concept is labeled HadoopOn Demand (HOD). The HOD approach utilizes the Torque resource manager for node allocation to size the virtual cluster. Within the virtual environment, the HOD system prepares the configuration files in an automated manner, and initializes the system based on the nodes that comprise the virtual cluster. Once initialized, the HOD virtual cluster can be used in a rather independent manner. A certain level of elasticity is build into HOD, as the system adapts to changing workload conditions. To illustrate, HOD automatically de-allocates nodes from the virtual cluster after detecting no active jobs over a certain time period. This shrinking behavior allows for the most efficient usage of the overall physical cluster assets.hadoop on demand is considered as a valuable option for deploying Hadoop clusters within a cloud infrastructure.

VII CONCLUSIONS

The additional studies executed on the Hadoop cluster showed that a rather large data set (high space density) is required for the discussed akNN methodology to operate efficiently and effectively. Scaling k in the benchmark sets disclosed a rather substantial processing cost increase that is mainly due to executing additional, larger MapReduce records in the S3 and S4 processing cycles. In a nutshell though, Hadoop'sMapReduce and HDFS infrastructure provides straightforward and robust techniques that operate on commodity (inexpensive) HW, and does provide high data availability features to the user community. For (very) large, unstructured datasets, Hadoop provides the IT performance infrastructure necessary to deliver results in a timely fashion.
These technical challenges must be addressed for efficient and fast processing of Big Data. The challenges include not just the obvious issues of scale, but also heterogeneity, lack of structure, error-handling, privacy, timeliness, provenance, and visualization, at all stages of the analysis pipeline from data acquisition to result interpretation. These technical challenges are common across a large variety of application domains, and therefore not costeffective to address in the context of one domain alone.

ACKNOWLEDGEMENT

I am very grateful to Mr. Ashish Kumar Sharma and Mr. Jony Birla Assistant Professor in department of CSE, for their support to write this paper.
I am also very thankful to Dr. NeetuSharma,the head of department of computer science in Ganga Insititute of technology and management for her motivation and support during the paper.

REFERENCE:

[1] S.VikramPhaneendra&E.Madhusudhan Reddy "Big Data- solutions for RDBMS problems- A survey" In 12th IEEE/IFIP Network Operations & Management Symposium.
[2] HDFS: http://hadoop.apache.org/hdfs/
[3] Hadoop: http://hadoop.apache.org/
[4] F. N. Afrati, J. D. Ullman, "Optimizing joins in a map-reduce environment".
[5] http://www.datanubes.com/mediac/HadoopArchPerfDHT.pdf.
[6] http://searchcloudcomputing.techtarget.com/definition/big-data-Big-D