Performance Enhancement of Data Processing using Multiple Intelligent Cache in Hadoop

K. Senthilkumar
PG Scholar
Department of Computer Science and Engineering
SRM University, Chennai, Tamilnadu, India

K. Satheeshkumar
PG Scholar
Department of Computer Science and Engineering
SRM University, Chennai, Tamilnadu, India

Dr. S. Chandrasekaran
Professor
Department of Computer Science and Engineering
SRM University, Chennai, Tamilnadu, India

Abstract— Every day, we create 2.5 quintillion bytes of data, out of 2.5 quintillion data 90% of the data in the world today has been created in the last two years alone. This data comes from everywhere; sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records and cell phone GPS signals to name a few. These largely generated data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process the data within a tolerable elapsed time .To access and handle such large dataset, distributed systems is an efficient mechanism. Hadoop is new tool introduced to process (Store/Process) such large datasets. Hadoop came up with two components viz. to store huge data (HDFS) and process the same using MapReduce. HDFS follows a cluster approach in order to store huge amounts of data; it is scalable and works on low commodity. It uses MapReduce framework to perform analysis and carry computations parellely on these large data sets. Hadoop follows the master/slave architecture decoupling system metadata and application data where metadata is stored on dedicated server NameNode and application data on DataNodes. In this project work, study was performed on Hadoop Architecture, behavior of file system and MapReduce in detail and concluded that processing of MapReduce is slow which was further confirmed by initial analysis and experiments performed on default Hadoop configuration. It is known that accessing data from caches much faster as compared to disk access. Multi Intelligent caching is one such mechanism in which the cache distributed over redis servers and this redis server (single place to store all cached data) serves client request. This mechanism helps in improving the performance, reducing access latency and increasing the throughput. In order to enhance and improve the performance of MapReduce, the new design of HDFS is proposed by introducing multi intelligent caching concept with redis server.

Keywords—Hadoop, HDFS, Map Reduce Framework, Hadoop Data Processing, Hadoop Caching

I. INTRODUCTION

Hadoop is a widely used distributed system, follows clustered approach, highly scalable and it allows massive amounts of data to be stored. Hadoop follows the master/slave architecture decoupling system metadata and application data where metadata is stored on dedicated server NameNode and application data on DataNodes. Over the years, Hadoop has gained importance because of its scalability, reliability, high throughput and performing analysis and large computations on these massive amounts of data. Currently, Hadoop is being used by all the leading industries like the Amazon, Google, Facebook, Yahoo etc. Hadoop's filesystem architecture and data computational paradigm has been inspired by Google File System and Google's MapReduce. At Yahoo, there is a of span 25,000 servers, and stores 25 petabytes of application data, with the largest cluster being 3500 servers. In a paper presented at Sigmod, describes how Facebook is using Hadoop in real time, with only few modifications made to it, it provides high throughput and low latency. It used in online travel, Mobile Data, E-Commerce, Energy Discovery etc... by various organizations across globe.

A. Data Storage in Hadoop

The Hadoop Distributed File System (HDFS) is the primary storage system used by Hadoop applications. HDFS is a distributed file system that provides high-performance access to data across Hadoop clusters. Like other Hadoop-related technologies, HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.

Because HDFS typically is deployed on low-cost commodity hardware, server failures are common. The file system is designed to be highly fault-tolerant, however, by facilitating the rapid transfer of data between

compute nodes and enabling Hadoop systems to continue running if a node fails. That decreases the risk of catastrophic failure, even in the event that numerous nodes fail.

When HDFS takes in data, it breaks the information down into separate pieces and distributes them to different nodes in a cluster, allowing for parallel processing. The file system also copies each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the others. As a result, the data on nodes that crash can be found elsewhere within a cluster, which allows processing to continue while the failure is resolved.

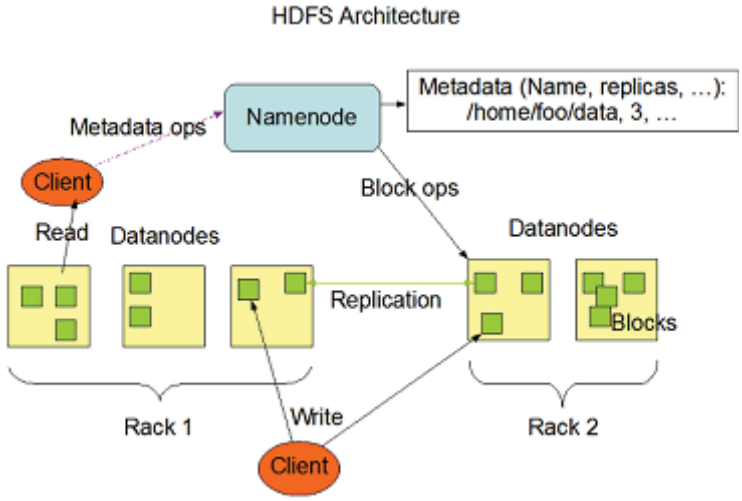

Figure 1. HDFS Architecture

HDFS is built to support applications with large data sets, including individual files that reach into the terabytes. It uses a master/slave architecture, with each cluster consisting of a single NameNode that manages file system operations and supporting DataNodes that manage data storage on individual compute nodes.

B. Data Analytics Using MR

Hadoop is using the Map Reduce framework to perform analytics and various computations in clustered environment. MapReduce is the heart of Hadoop. It is this programming paradigm that allows for massive scalability across hundreds or thousands of servers in a Hadoop cluster. The MapReduce concept is fairly simple to understand for those who are familiar with clustered scale-out data processing solutions.

The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform. The first is the map job, which takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce job is always performed after the map job.

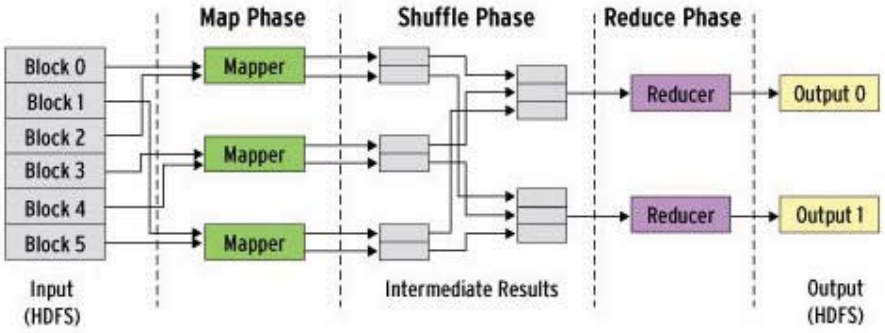

Figure 2. Map Reduce Framework

Map Phase in Map Reduce framework is reading the metadata information from disk which is I/O intensive task. In order to improve the performance of this Map Reduce task this need optimization. Recently Cloudra published paper “Optimizing Map Reduce Performance” which states that the performance of MR is slow which need optimization in order to reduce task execution time. In depth study of MR mentions that Map Reduce is

slow, the processing time can be improved by adding more data nodes into the HDFS cluster. But adding more data nodes is not a cost effective solution for big clusters (more than 600 nodes). During initial phase of Map Reduce the data was being read from disk, there were more local task scheduled.

II. Related Work

According to PACMan, when multiple jobs are run in parallel, job's running time can be decreased only when all the inputs related to running a job are cached. According to Dhruba, either cache all the inputs related to that particular job or do not cache the inputs at all. Caching only part of the inputs will not help in improving the performance. These massive distributed clustered systems have large memories and job execution performance can be improved if these memories can be utilized to the fullest. PACMan is a caching service that coordinates access to the distributed caches. This service aims at minimizing total execution time of job by evicting those items whose inputs are not completely cached. For evicting the inputs which have been minimally used, LFU-F algorithm, LIFE sticky policy has been proposed. They define a new parameter wave-width of the job which refers to total tasks which can be executed in parallel at a time. They have used MapReduce and Dyrad as examples to illustrate their algorithm and hypothesis. According to them reading the raw input from filesystem is IO intensive and forms 79% of the phase. They conducted experiments on Hadoop and observed improvement in job execution time. They emphasize on memory-locality tasks which is an important factor is contributing to cluster efficiency. Dhruba et.al proposes an architecture called the PACMan which coordinates the caches globally and it takes care of two things which is support queries where block is cached and coordinating the cache replacement. Its architecture includes a coordinator service and PACManClients are located on nodes where data lies. Blocks are cached on these clients. The coordinator includes the information regarding this block belonging to which file and wave-width of the file. This overall structure of the coordinator is used for scheduling tasks which are memory local, in implementing sticky policy and to check on the incomplete files. If the data is not cached then it accesses disk. Also it emphasizes to schedule a data local job. For replacement policies they go for global cache replacement policies which are LIFE and LFU-F. The overall job execution time is reduced attempting to schedule jobs of smaller-wave widths. This system does not take into account the remote caching.

A. Dynamic Caching

The main aim defined in paper Dynamic Caching is caching mechanism allowing concurrent access to data and proposes algorithms relating to locality of data which focuses on the decrease in the overall job completion time. For implementing Dynamic caching they are using Hadoop and their caching mechanism is based on Memcached which are a set servers storing the mapping of block-id to datanode-ids. These servers also serve the remote cache requests. The caching of blocks is carried out on DataNodes. The paper mentions about two different design architectures, First architecture defines; to serve the request of DataNode, simultaneous requests are sent to NameNode and Memcached servers. DataNode receives reply from both of them, but it checks if true from Memcache then access block from Memacache else access the block from disk whose location as indicated by NameNode. In second design architecture, again simultaneous request is sent to Memcache and NameNode, but NameNode does not reply back until indicated by Memcache about unavailability of block where in such a case NameNode sends block locations to DataNode. The design includes prefetching where whenever request is seen by Memcache, neighbouring blocks are also looked up. If neighbouring blocks are missing then Memcache requests NameNode to look for replicas and if available, requests are sent to DataNode to cache blocks and Memcache updates it's locations. The caching system designed is not distributed and for lookups it is always required to contact the single set of Memcache servers. Also there incurs an extra delay with respect to second architecture when Memcache does not contain the blocks in cache.

B. Hadoop Collaborative Caching

Hadoop Collaborative Caching is a caching mechanism which is maintaining local cache, remote cache where data node can access the data from Local cache, if it's not available look into remote cache. Totally two modules in Hadoop CC Function Manager, Cache Manager. Function Manager handles request from client and look for metadata cache and return the cached block list back to client. This operation happens in NameNode. Cache Manager resides in Data Node which serves client request. Cache manager will maintain different list of cache frequent cache, Recent cache, History cache. Name node is single point of contact for entire cluster system, function manager reside in Name Node and serves client. Caching all metadata is required good amount of memory in name node and also other threads requires memory to perform their own task. There is no automatic eviction policy in this function & cache manager. Whenever new entry added into to cache list it's more important to check the available memory in data node. Different tasks like Performing Computation of available memory, adding new entry into cache list, eviction of existing entries are needed in this Hadoop Collaborative Caching architecture.

III. Proposed Work

A. Hadoop Redis Caching

Accessing data from cache is much faster as compared to disk access. Hence to improve the performance of MapReduce jobs and improve the overall cluster efficiency of Hadoop system, improvements and architectural changes were incorporated in Hadoop Distributed filesystem which led to new system called Hadoop-RCaching (Hadoop Redis Caching). Hadoop R Caching is one such mechanism in which the cache distributed over dedicated server form a single cache to serve the requests. Redis act as a Cache Server where other components of system make use of it to store their individual cache information.

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets. You can run atomic operations on these types, like appending to a string; incrementing the value in a hash; pushing to a list; computing set intersection, union and difference; or getting the member with highest ranking in a sorted set. It's key value store & in memory storage system so that retrieval of data is much faster as compared to other system.

Redis as a cache is the max memory directive, a feature that allows specifying a maximum amount of memory to use. When new data is added to the server, and the memory limit was already reached, the server will remove some old data deleting a volatile key, that is, a key with an EXPIRE (a timeout) set, even if the key is still far from expiring automatically.

B. Components of Hadoop –RCaching

The components of Hadoop-RCaching are,

- **RGCM** – Redis Global Cache Manager
- **RLCM** – Redis Local Cache Manager
- **R**-Server

Redis Global Cache Instance is mainly to store and retrieve global cache information FSI image file will be loaded into memory and act as a cache layer. For each update in FSI image will be captured and pushed to RGCM instance to keep the updated global cache.

Caches of all the participating DataNodes machines taken together formed a single cache or global cache. NameNode is the central co-ordinator of this global cache, but allowing the decisions of remote caching to be taken by RLC Instance on DataNodes. New Approach allows efficient use of resources where instead of increasing the number of nodes, more slots can be added in order to improve performance. Caching of data was made faster and easier by the reference caching technique.

A HDFS block is composed of two files which is metafile and the block file. Meta file refers to checksum value of the data and block file refers to actual data. If these file references are cached, it helps in locating the meta files faster for checksum checks, faster caching of data from disk to memory since not much time is spent in searching for files when data stored is about petabytes. This Redis in memory caching mechanism provides an additional method of reducing the overall time.

IV. System Architecture

Below section explains the various components of Hadoop –RCaching, Components added into existing core Hadoop architecture to enhance performance of data processing.

A. RGC Manager

NameNode have their own Redis Global Cache Manager instance in order to access the metadata cache when DFS client request for blocks. It receives request from client and access the Redis Global Cache Manager Instance to retrieve the list of cached blocks for that particular block. Name Node access this Redis Global cache instance to add new block report in case of change in block report.

B. RLC Manager

Each of Data Node will access the Redis Local Cache Manager which is responsible for managing caches, look up in local as well as in global cache image upon request from DFS client. If data node not found the cache in RCL Instance it will look up Global Cache Image to check if cache meta data available or not.. In absence of cached block from both Local & Global Redis Instance data node en queue the entry into local cache. Data Node will send the block report to name node, this case name node will update or add block into global instance.

Blocks in the local instance will be replaced when cache is fully utilized based on Redis Server configuration.

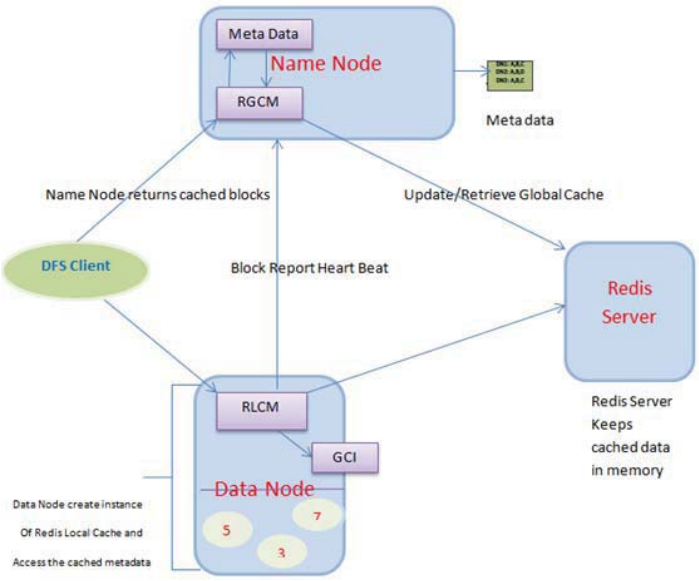

Figure 3. System Architecture

C. Name Node

NameNode is the central co-ordinator for maintaining the Global Cache Image. It builds its global cache image when it obtains the cached block report from the DataNodes. As soon as it obtains its report, it updates the mapping of cached block to DataNode. Upon updation of the Global Cache Image, as a response to cached block report it sends a copy of Global Cache Image to the DataNode.

D. Data Node

DataNode provides a cached block report of its local cache to NameNode after periodic interval. As a response to this report NameNode commands DataNode to update global cache image.

E. DFS Client

DFSClient receives set of caching DataNodes along with non-caching DataNodes from NameNode as response to its request to read a particular file.

F. Redis (Cache Server)

Redis as a cache used to work in two main ways: you could either set a time to live to cached entries. If you tune the TTL well enough, and you know how many new objects are created every second, you can avoid Redis using more than a given amount of RAM. Another way to use Redis as a cache is the maxmemory directive, a feature that allows specifying a maximum amount of memory to use. When new data is added to the server, and the memory limit was already reached, the server will remove some old data deleting a volatile key, that is, a key with an EXPIRE (a timeout) set, even if the key is still far from expiring automatically.

The algorithm used is very simple, three random volatile keys are sampled, the key with the nearest expire time is removed from the dataset. If there are not volatile keys at all the server returns an error, as a write operation was requested, we are already at the memory limit specified by the user, and no volatile keys can be removed to make room for new data.

G. Configuration Parameters

1) maxmemory bytes: This works as usually, specifying the max number of bytes to use. You can specify this as kbytes, gigabytes and so forth as usually, like maxmemory 2g.

2) maxmemory-policy policy: This new configuration option is used to specify the algorithm (policy) to use when we need to reclaim memory. There are five different algorithms now:

a) volatile-lru remove a key among the ones with an expire set, trying to remove keys not recently used.

b) volatile-ttl remove a key among the ones with an expire set, trying to remove keys with short remaining time to live.

c) volatile-random remove a random key among the ones with an expire set.

d) allkeys-lru like volatile-lru, but will remove every kind of key, both normal keys or keys with an expire set.

e) allkeys-random like volatile-random, but will remove every kind of keys, both normal keys and keys with an expire set.

Redis is a single-threaded server. This means that a single thread reads incoming connections using an event-based paradigm such as epoll, kqueue and select. When a specific event occurs on a file descriptor, it processes them and write back responses.

V. Performance Analysis

A. Core Hadoop Improvements

The proposal to implement Redis and integrating with Hadoop works. It will improve the Map Reduce Job Performance by lowering the job execution time which led to improvement and enhancement in the overall system. Reading data from cache is always faster as compared to reading data from the disk. On the DataNode side, the data was being streamed from disk hence the overall performance of a MapReduce job was considerably slow and had a overall high I/O rate. An attempt has been made to cache this data and stream from cache. Moreover in memory Redis is faster retrieval and helps to access the cached data in Data Node quickly in order to enhance the performance of Hadoop.

Caching of input data at the DataNode level helps in improving job execution time. A distributed cache structure is followed where the DataNodes have common cache which prevents to search it from other Data Node cache. We are utilizing the memories of all the participating DataNodes thus reducing the no. of disk accesses. If the data is not found in requested node's cache but found in global image cache, so instead of serving from disk we are serving from cache and which saves us execution time. This approach not only reduces job execution time, but also helps us to utilize the resources efficiently. To improve the performance, instead of adding more nodes, we can focus on adding more slots causing more map and reduce tasks to be scheduled at once parallely and with the technique of Hadoop RCaching, data is available in cache causing in overall lowering of the job execution timing.

Whenever JobClient submits the Job, JobTracker tries to schedule the job on the same node as the input. More the data-local tasks, better the execution time. So as soon as TaskTracker contacts JobTracker either with slot available or no, if the slot is available then schedule a task. If the input required for the task resides on a different node, then it is rack-local and in such a case the data is streamed from other node to the node where the task is scheduled and then the task is carried out. This results in added extra time to complete the task resulting in increasing the overall job execution time. But with caching, the tasks scheduled get completed earlier and slots become available faster which provides room for tasks to be scheduled as data-local, hence improving the overall execution time. Also in terms of caching, the data is streamed from neighbouring node's cache, although there is n/w I/O involved, but minimal and moreover, it is from cache hence it reduces the overall time.

VI. Conclusion

A new architecture Hadoop RCaching aimed to enhance the data processing in Hadoop using multi intelligent (Local, Remote) caching. Data is served from local caches as well as remote caching. Integration with Redis in memory caching helped to evict the old cache automatically based on Time to live component set during the Redis Server Setup and maintains replication copy of cached data for high availability. Name Node and Data Node making use of Redis Server in order to access cached data faster, Hence overall job execution time reduced by considerable amount, efficiency of the system increased

References

[1] G. Ananthanarayanan, A. Ghodsi, A.Wang, D.Borthakur, S.Kandula and I. Stoica."PACMan: Coordinated memory caching for parallel jobs", In NSDI, 2012.

[2] http://hadoop.apache.org/docs/r0.20.2/hdfs_design.html

[3] Meenakshi Shrivastava, "Hadoop-Collaborative Caching in Real Time HDFS", Dec 2012, Rochester Institute of Technology, Rochester, NY, USA

[4] http://en.wikipedia.org/wiki/Apache_Hadoop

[5] http://www.slideshare.net/cloudera/mr-perf

[6] http://developer.yahoo.com/hadoop/tutorial/module4.html

[7] http://bradhedlund.com/2011/09/10/understanding-hadoop-clustersand-the-network/

[8] http://cleversoft.wordpress.com/2012/12/10/redis-server-deployment-guide/

[9] http://oldblog.antirez.com/post/redis-as-LRU-cache.html

[10] http://ruturaj.net/redis-memcached-tokyo-tyrant-and-mysql-comparision/

[11] http://stackoverflow.com/questions/2873249/is-memcached-a-dinosaur-in-comparison-to-redis

[12] http://www.enjoythearchitecture.com/redis-architecture