IJETIE International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

SYNCHRONIZED PROCESS BASED SCHEDULING TO IMPROVE MAPREDUCE EXECUTION STRATEGY

S.Preetha
PG Scholar[1]
Computer Science and Engineering,
P.S.R.Engineering College,
Sivaksi.
s.preethas.preetha@gmail.com

R.Ramani
Research Scholar[2]
Computer Science and Engineering,
P.S.R.Engineering College,
Sivaksi.
rramani.ananth@gmail.com

ABSTRACT
MapReduce is an effective programming model for large scale data processing. Hadoop is a distributed batch processing infrastructure which is currently being used for big data management. The foundation of Hadoop consists of Hadoop Distributed File System (HDFS). HDFS presents a client-server architecture comprised of a NameNode and many DataNodes. The NameNode is dedicated to maintain the file system metadata consisting of files, directories, and blocks, controls the work of other cluster nodes, DataNodes, handling the actual data blocks of files.The system is designed to scale linearly on the number of DataNodes, as they can process data transfers independently of each other. The NameNode holds file system metadata in memory, and thus the limit to the number of files in a file system is governed by the amount of memory on the NameNode. Thus when the memory on NameNode is full there is no further chance of increasing the cluster capacity. The goal of this paper used the concept of cache memory for handling the issue of NameNode scalability.

General Terms
Scheduling Algorithm.

Keywords
Hadoop, NameNode, DataNode, HDFS, Cache.

1. INTRODUCTION

Hadoop is a Large scale, open source software framework. First developed and released as open source by Yahoo, it implements the MapReduce approach pioneered by Google in compiling its search indexes. To store data, Hadoop utilizes its own distributed filesystem, HDFS, which makes data available to multiple computing nodes. A typical Hadoop usage pattern involves three stages namely, loading data into HDFS,MapReduce operations, and retrieving results from HDFS.

Hadoop Distributed File System (HDFS) is designed to reliably store very large files across machines in a large cluster. It is inspired by the Google File System. Hadoop DFS stores each file as a sequence of blocks, all blocks in a file except the last block are the same size. Blocks belonging to a file are replicated for fault tolerance. The block size and replication factor are configurable per file. Files in HDFS are "write once" and “read many times” but strictly one writer at any time.

Like Hadoop Map/Reduce, HDFS follows a master/slave architecture. An HDFS installation consists of Name Node, Data Node, Secondary Name Node, Job Tracker ,Task Tracker. The NameNode acts as a master server that manages the filesystem namespace and regulates access to files by clients. The Namenode makes filesystem namespace operations like opening, closing, renaming etc. of files and directories available via an RPC interface. There are a number of Data Nodes**,** one per node in the cluster. The Datanodes are responsible for serving read and write requests from filesystem clients, they also perform block creation, deletion, and replication upon instruction from the Namenode.

The NameNode keeps a reference to every file and block in the filesystem in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling [7]. Thus when a NameNode memory is full it becomes difficult to just add another node in the cluster for further storage needs as the NameNode will not be able to handle this extra node and filesystem metadata that will be generated from it. This raises the NameNode scalability issue.

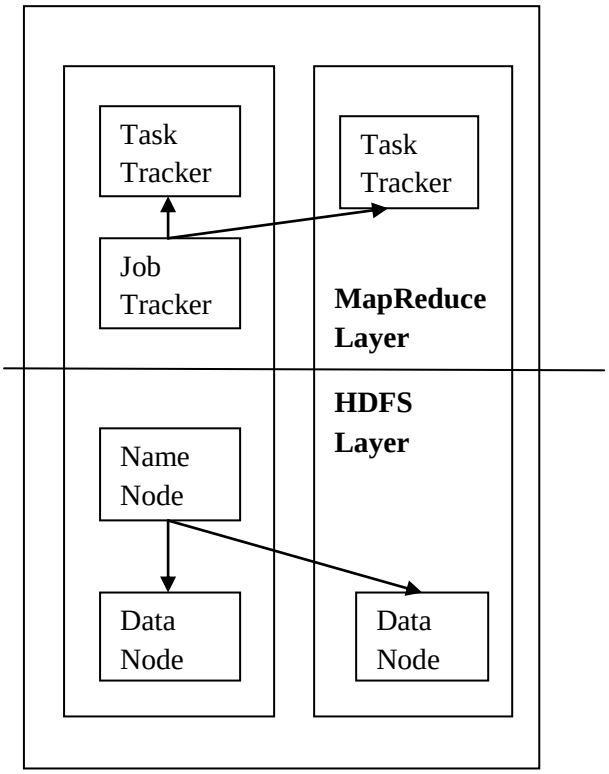
The goal of this paper is to provide a solution to reduce the use of NameNode memory space so that the problem of NameNode memory getting full which ultimately results in hindering the scalability of cluster will be delayed.

IJETIE
International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

2. HADOOP ARCHITECTURE

Hadoop is a distributed batch processing infrastructure which is currently being used for big data management. Hadoop is designed to be parallel and resilient. It redefines the way that data is managed and processed by leveraging the power of computing resources composed of commodity hardware and it can automatically recover from failures.

The basic Hadoop architecture consists of two primary components viz. 1. Hadoop Distributed File System (HDFS) and 2. MapReduce.



2.1 Hadoop Distributed File System (HDFS layer)

Hadoop Distributed File System (HDFS) serves as the large scale data storage system. Similar to other common file systems, the HDFS supports hierarchical file organization. The NameNode splits large files into fixed sized data blocks which are scattered across the cluster. Typically the data block size for the HDFS is configured as 128MB, but it can be configured by file system clients as per usage requirements. The data storage is of type write once/read many (WORM) and once written, the files can only be appended and cannot be modified to maintain data coherency. Since HDFS is built on commodity hardware, the machine failure rate is high. In order to make the system fault tolerant, data blocks are replicated across multiple DataNodes. HDFS provides replication, fault detection and automatic data block recovery to maintain seamless storage access. By default replication takes place on three nodes across the cluster . When a client tries to access the failed DataNode, the NameNode maps the block replica and returns it to the client. For achieving the high throughput, the file system nodes are connected by high bandwidth network.

2.2 NameNode

The NameNode maintains the file system metadata as the HDFS directory tree and operates as a centralized service in the cluster. It controls the mapping between file name, data block locations and the DataNodes on which data blocks are stored. It also writes the transaction logs to record modifications in the file system. Clients communicate with the NameNode for common file system operations such as open, close, rename and delete.

2.3 DataNode

A DataNode is a storage server that accepts read/write requests from the NameNode. DataNodes store data blocks for local or remote clients of HDFS. Each data block is saved as a separate file in the local file system of the DataNode. The DataNode also performs block creation, deletion and replication as a part of file system operations. For keeping the records up-to-date, the DataNode periodically reports all of its data block information to the NameNode. 9 DataNode instances can talk to each other for data replication. To maintain its live status in the cluster, it periodically sends heartbeat signals to the NameNode. When the NameNode fails to receive heartbeat signals from the DataNode, it is marked as a dead node in the cluster.

3. EXISTING SYSTEM

The NameNode server in the Hadoop cluster keeps the track of filesystem metadata, it keeps a track of how your files are broken down into file blocks, which nodes store those blocks, and the overall health of the distributed filesystem. It maintains a catalog of all block location in the cluster which makes the NameNode the bookkeeper of HDFS.

Since the NameNode is a single container of the file system metadata, it naturally becomes a limiting factor for file system growth. In order to make metadata operations fast, the NameNode loads the whole namespace into its memory, and therefore the size of the namespace is limited by the amount of RAM available to the NameNode [9]. Thus when the memory available to the NameNode is full there are no chances for the cluster to grow limiting the number of active clients. Also a big HDFS installation with a NameNode operating in a large JVM where the memory space is almost full is vulnerable to frequent full garbage collections, which may take the NameNode out of

IJETIE International Journal of Emerging Technology and Innovative Engineering
Volume I, Issue 4, April 2015
ISSN: 2394 - 6598
www.ijetie.org

service for several minutes [11]. Thus it is clear that memory available to the NameNode machine in a Hadoop cluster dictates the size of cluster and ultimately the number of active clients using a Hadoop based application.

4. PROPOSED WORK

4.1 The basic idea of cache memory

A computer has a wide range of type, technology, performance and cost when it comes to complex memory management. Also a computer memory has a hierarchy from Central Processing Unit (CPU) registers to magnetic taped. Cache is one such memory type which comes second in the hierarchy which is an expensive yet fast access data storage device. A cache in generic terms is an intermediate buffer memory used to reduce the average time to access data. The cache is a smaller, faster memory which stores copies of the data from frequently used main memory locations. Cache memories are used in modern, medium and high-speed CPUs to hold temporarily those portions of the contents of main memory which are currently in use. Information located in cache memory may be accessed in much less time than that located in main memory. Thus, a central processing unit (CPU) with a cache memory needs to spend far less time waiting for instructions and operands to be fetched and/or stored. The instance when the main memory has to be accessed is only when data to be fetched is not in the cache i.e. a cache fault. Such cases arise very less number of times compared to cache hit.

4.2 Design Overview

The basic concept of cache i.e. storing the frequently used data closer to the processor than the whole data and using it for faster operation can be applied in the NameNode memory management. Thus when the NameNode is operating in a heavy HDFS installation, the metadata records of frequently used data that is being used by most of the active clients can be kept in the NameNode memory space and the metadata records of least used data which sometimes can even be irrelevant can be kept at a different location and accessed as and when required. This way a lot of NameNode memory usage will be reduced which then can further be used to store more frequent data. Also with a less loaded HDFS NameNode the JVM garbage collection will be reduced which will avoid the problem of NameNode becoming irresponsive due to the excessive garbage collection.

4.3 Scheduling Algorithm

The scheduling algorithm will be used for moving the least recently used metadata from NameNode to secondary storage device. The scheduling algorithm takes into account two fields from the metadata records two decide whether or not to remove any metadata record. By default each metadata record keeps a track on last access time. Each time the access time is updated the count is increased by one indicating that the data/file represented by it is been used. Once triggered the scheduling algorithm works as follows:

1. Initially a mean of count which is the frequency of use of a file is calculated.
2. For each metadata record
   a. First check the last access time, if it indicates that the file has been accessed recently, then the metadata record is not removed irrespective of whether the file is used more frequently.
   b. If the access time indicates that the file is not been used for long time then the count i.e. the frequency of use of file is compared against the mean count that is calculated in the first step.
   c. If the count field in the metadata record is greater than the mean, suggesting that the data is used more frequently though it was not used recently. This metadata record is not removed.
   d. Rest of the metadata records are separated out and stored into another file viz. fsimage2.

This scheduling policy removes near about 30% of the metadata records from the fsimage file thus making space available for other purposes.

In the proposed work, a threshold value will be defined on the NameNode memory space. As and when this threshold value will be reached the scheduling algorithm will run so as to remove the metadata records that are not being used. Initially the NameNode will keep storing the filesystem metadata as it comes until the predefined threshold is reached. Once the threshold will be reached the scheduling algorithm will come into play and remove some of the records based on the scheduling function. This separated data will be moved to secondary storage device.

When the client performs a read operation the usual processing that is done by the NameNode to fetch metadata for requested file will be done and the client will be given back the file. The only difference will occur when the requested file is moved to the secondary storage as it was not recently used and the NameNode has reached threshold value. In this case the NameNode will not find the record in its memory and thus a request will be made to secondary device to fetch the metadata record and the metadata record will be removed from secondary device and will be loaded again on the RAM.

5. CONCLUSIONS

Thus a way to reduce the NameNode memory consumption thus increasing the capacity of a Hadoop cluster. By implementing the cache concept in HDFS the two issues with Hadoop viz. NameNode being

irresponsive due to garbage collection and cluster scalability issue can be solved. This way the Hadoop cluster will not reach the stage where the NameNode becomes irresponsive due to excessive JVM garbage collection as the HDFS will not be heavily loaded. Also as the NameNode will only store relatively more frequently used data the operations carried on the cluster will be faster and more efficient.

6 REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters", Google Research Publication, OSDI"04: 6th Symposium on Operating Systems Design and Implementation, pp. 137-149, 2004.

[2] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The Hadoop Distributed File System", IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), Sunnyvale, California USA, vol. 10, pp. 1-10, 2010.

[3] T. White, "Hadoop: The Definitive Guide", O"Reilly, Scbastopol, California, 2009.

[4] R. Buyya, J. Broberg, A. Goscinski, "Cloud Computing: Principles and Pradigms", Wiley, 2011.

[5] C. Lam, "Hadoop in Action", Manning, 2011.

[6] L. Jing-min, H. Guo-hui, "Research of Distributed Database System Based on Hadoop", IEEE International conference on Information Science and Engineering (ICISE), pp. 1417-1420, 2010.

[7] MapReduce, http://en.wikipedia.org/wiki/MapReduce.

[8] Konstantin V. Shvachko "HDFS scalability: The limits to growth". ; LOGIN: VOL. 35, NO. 2.

[9] Alex Holmes, "Hadoop in Practice", Manning, 2011

[10] Konstantin V. Shvachko "Apache Hadoop: The Scalability Update"; login: JUNE 2011 Apache Hadoop.

[11] Kyong-Ha Lee, Yoon Joon, Lee, Hyunsik Choi, Yon Dohn Chung, Bongki Moon "Parallel Data Processing with MapReduce: A Survey".

[12] Y. Pingle, V. Kohli, S. Kamat, N. Poladia, "Big Data Processing using Apache Hadoop in Cloud System", National Conference on Emerging Trends in Engineering & Technology, pp. 475-479, 2012.

[13] Apache-Hadoop, http://Hadoop.apache.org