International Journal of Advance Research in Science and Engineering
Volume No.06, Special Issue No.(01), December 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

# Native Hadoop Based Enhanced Cloud Architecture in biomedical industry

**K. Arul Jothy[#1], K. Sivakumar[*2], Dr.G.Rajiv Suresh Kumar[#3]**

*[1]Final Year CSE, [2]Assistant Professor, [3] Professor*

*Department of Computer Science and Engineering,*

*JCT College of Engineering and Technology,*

*Coimbatore, (India)*

**ABSTRACT**

*Explosion of biological data due to large scale genomic research and advances in high throughput data generation tools result in massive distributed datasets. Analysis of such large non relational, heterogeneous, and distributed datasets is emerging challenge in data driven biomedical industries. Highly complex biological data require unconventional computational approaches and knowledge-based solutions. Distributed datasets need to be reduced to smaller datasets that can be efficiently queried. Since genomic and biological data is generated in large volume and is stored in geographically diverse locations, distributed computing on multiple clusters, our objective here is to assess the feasibility of using Cloud based platform to analyze genomic big data. In this paper we report on the limitation of cloud based platform in the analysis of genomic data and we implement the edtc3 with native hadoop infrastructure to overcome its limitations.*

***Keywords: Cluster, Cloud, Distributed Computing, EDTC3, Native Hadoop***

## I. INTRODUCTION

Bioinformatics applications usually require large complex amounts of data processing and computational capabilities. A large distributed file based processing is adopted in this project to process large data files which can scale up to few terabytes. Native Hadoop based cloud architecture is composed of Hadoop Distributed File System (HDFS) (Fig.2), MapReduce programming model and etcd3 as (Fig.1) coordination service. HDFS cluster is composed of a centralized indexing system called NameNode and its data processing units called DataNodes; together they form a unique distributed file system. NameNode plays an important part in supporting the Hadoop Distributed File System by maintaining a File-Based block index map, this map is responsible to locate all the blocks related to the HDFS. HDFS is the primary storage system; HDFS creates multiple replicas of data blocks and is further responsible to distributes data blocks throughout a cluster to enable reliable, extremely rapid computations [1]. Etcd3 is critical component of the infrastructure, it provide coordination and messaging across applications [2]. The Etcd3capabilities include

naming, distributed synchronization, and group services.And Etcd3 provides a distributed key-value store. Etcd3 is guarantees in providing the sequential consistency and serializable isolation. Hadoop framework leverages on large-scale data analysis by allocating data-blocks among distributed DataNodes.

Hadoop Distributed File System allows the distribution of the data set into many machines across the network that can be logically combined for processing.

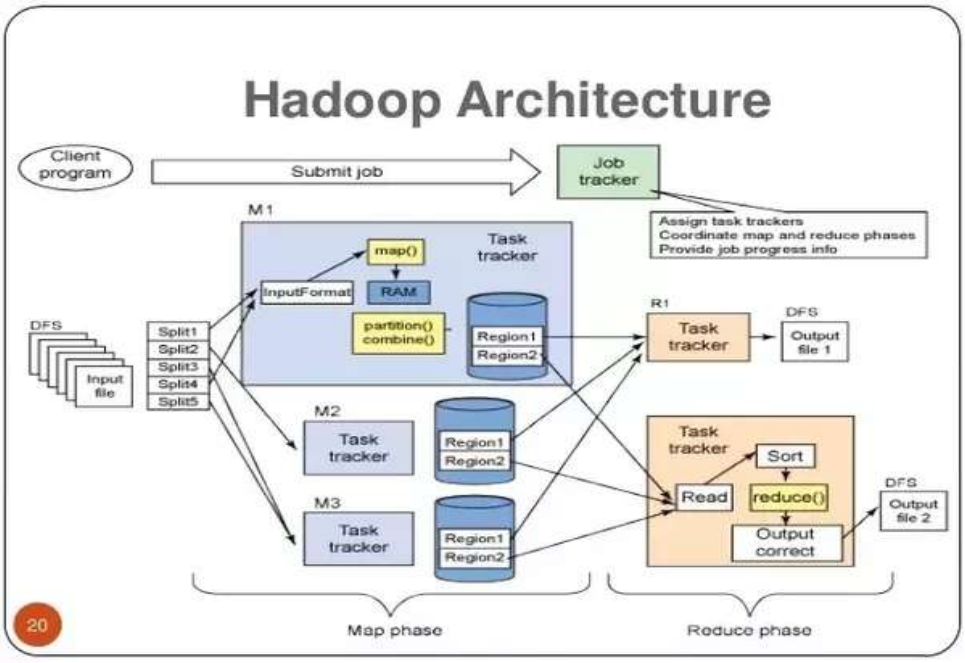


**Fig 1: Hadoop Architecture**

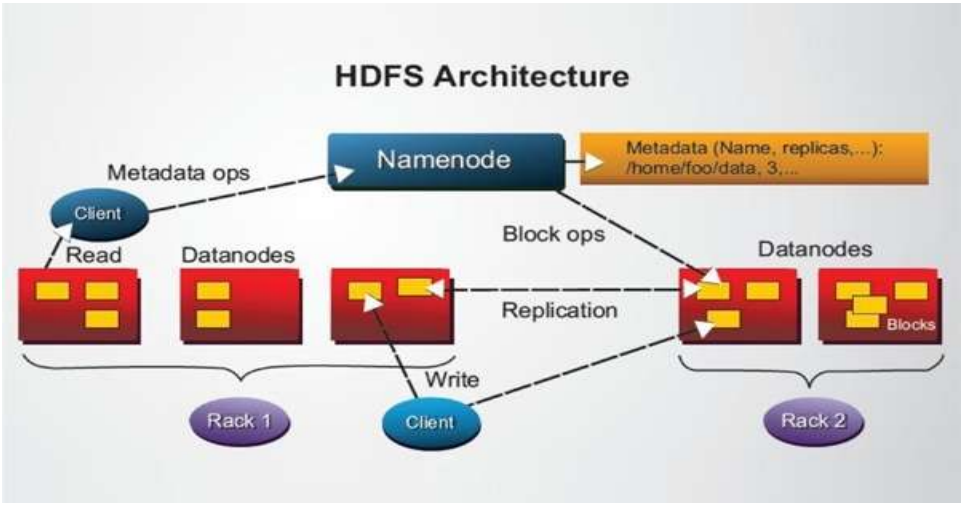


**Fig 2: HDFS Architecture**

International Journal of Advance Research in Science and Engineering
Volume No.06, Special Issue No.(01), December 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

HDFS adopted Write-Once-Read-Many model to store data in distributed DataNodes. NameNode is responsible for maintaining namespace hierarchy, managing datablocks and DataNodes mapping. Once job information is received from the client, NameNode provides a list of available data nodes for the job. NameNode maintains the list of available data nodes and is responsible to update the index list when a DataNode is unavailable of failed due to hardware or network issues. A heartbeat is maintained between the NameNode and the DataNodes to check the keep-alive status and health of the HDFS. Client writes data directly to the DataNode (fig 3). HDFS is architected to have the block fault and replication tolerance. NameNode is responsible to maintain a healthy balance between disk processing on various DataNodes and has an ability to restore the failed operations on the remote blocks. Data Locality is achieved through cloud file distribution, the file processing is done local to each machine, and any failed reads from the blocks are recovered through block replication. The process of selecting the mappers and the reducers is done by the JobTracker immediately after lunching a job [3, 4].

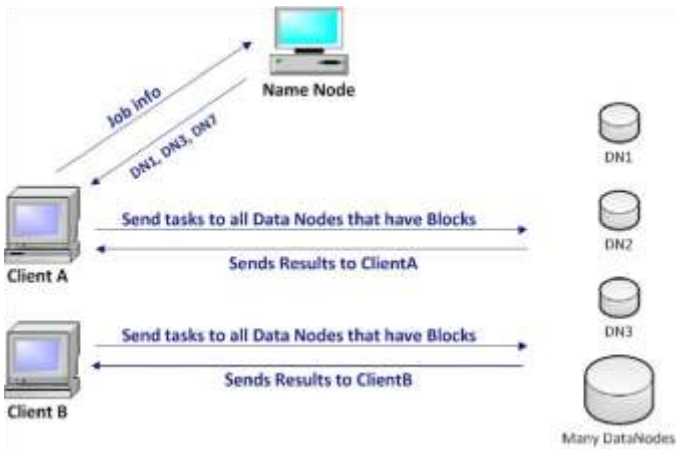


**Fig. 3: DataNodes and Task Assignment**

A client operating on the HDFS has network file transparency and the distribution of blocks on different machines across the Cloud is transparent to the client. HDFS is oriented towards hardware transparency. Processing DataNodes can be commissioned or decommissioned dynamically without affecting the client.

## II. NATIVE HADOOP AND BIOIFNRMATICS DATA

It is also important to realize that any bioinformatics tools such as (BLAST, FASTA etc) can be processed in parallel, but most of the users are not trained to modify the existing applications to incorporate parallelism effectively [5]. This project leverages on HDFS distributed computing model utilizing various commodity servers and Hadoop Apache Map-Reduce frameworks to explore genome data.

**Example: finiding a Specific Pattern Sequence in DNA Genome**

1) Background:

Massive genomic data, driven by unprecedented technological advances in genomic technologies, have made genomics a computational research area. Next generation sequencing (NGS) technologies produce high throughput short read (HTSR) data at a lower cost [6]. Scientists are using innovative computational tools that allow them rapid and efficient data analysis [7, 8] . DNA genome sequence consists of 24 chromosomes. The compositions of nucleotides in genomic data determine various traits such as personality, habits, and inheritance characteristics of species [9]. Finding sequences, similarities in sequences, subsequences or mutation of sequences are important research area in genomic and bioinformatics. Scientists need to find subsequences within chromosomes to determine either some diseases or proteins frequently. Each chromosome has many known genes and many unknown sequences. For example, chromosome one consists of about 249 million of nucleotide base pairs, which represent about %8 of the total DNA in human cells. The total number of genes in chromosome 1 is about 4,316 genes each one has different length of base pairs.

2) The problem definition and solution using Hadoop

Searching for sequences or mutation of sequences in a large unstructured dataset can be both time consuming and expensive. Sequence alignment algorithms are often used to align multiple sequences. Due to memory limitation, aligning more than three – four sequences is often not allowed by traditional alignment tools.

In this project we proposed suing HadoopMapReduce to align genomic data. We tested MapReduce for sequence alignment by building a complete MapReduce program that takes the pattern sequence as a key and find this sequence in a chromosome and also in the whole DNA sequence. We executed the job within a cluster that has three DataNodes.

As expected Hadoop cluster with three nodes was able to search human genome much faster than single node, it is expected that search time will reduce as number of DataNodes are increased in the cluster.

## III. LIMITTAION OF HADOOP

Many Big Data problems, especially genomic data, deal with similarities/sequences and sub-sequences searches. If a “sub-sequence” is found in a specific blocks in a DataNode, sequence containing that subsequence can only exist in the same DataNode. Since current Hadoop Framework does not support caching of data, it ignores location of DataNode with sub-sequence and read data from all DataNodes for every new job [10]. Shown in Figure 2 Client A and Client B are searching for similar sequence in BigData. Once Client A finds the sequence, Clint B will also go through the whole BigData again to find the same results. Since each job is independent clients do not share results. Any client looking for Super sequence with sub-sequence already searched will have to go through the BigData again. Thus the cost to perform the same job will stay the same each time.

International Journal of Advance Research in Science and Engineering
Volume No.06, Special Issue No.(01), December 2017
www.ijarse.com
IJARSE
ISSN: 2319-8354

## IV.CURRENT ARCHITECTURE

### 1. NATIVE HADOOP

In current HadoopMapReduce architecture, the client first sends a job to the cluster administrator, which is the NameNode. The job can be sent either using Hadoop ecosystem (Query language such as Hive) or by writing a job source code [19]. Before that, the data source files should be uploaded to the HDFS by dividing the BigData into blocks that have the same size of data, usually 64 or 128 MB for each block. Then, these blocks are distributed among different DataNodes within the cluster. Any job now has to have the name of the data file in HDFS, the source file of MapReduce code (e.g. Java file), and the name of the file where the results will be stored in.

Native Hadoop architecture follows the concept of "write-once and read-many," so there is no ability to make any changes in the data source files in HDFS. Each job has the ability to access the data from all blocks. Therefore network bandwidth and latency is not a limitation in the dedicated cloud, where data is written once and read many times. Many iterative computations utilize the architecture efficiently as the computations need to pass over the same data many times.

Several research groups have also presented solutions about data locality to address the issue of latency while reading data from DataNodes [20]. Hadoop falls short of query optimization and reliability of conventional database systems.

In the existing HadoopMapReduce architecture, multiple jobs with the same data set work completely independent of each other. We also noticed that searching for the same sequence of characters. For example in any text format data requires the same amount of time each time we execute the same job. Also, searching for the supersequence of a sequence that has already been searched requires the same amount of time.

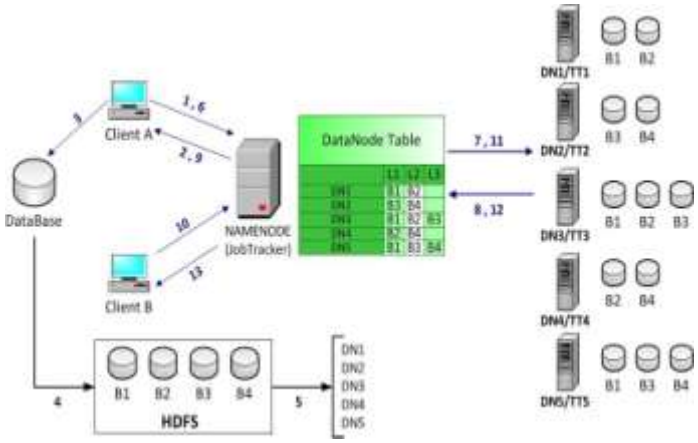


**Fig 4: Native Hadoop MapReduce Workflow**

## 2. NATIVE HADOOPMAPREDUCE WORKFLOW

**MapReduce workflow in native Hadoop has been explained in figure 4 as follows:**

Step 1: Client " A" sends a request to NameNode. The request includes the need to copy the data files to DataNodes.

Step 2: NameNode replays with the IP address of DataNodes. In the above diagram NameNode replies with the IP address of five nodes (DN1 to DN5).

Step 3: Client " A" accesses the raw data for manipulation in Hadoop.

Step 4: Client "A" formats the raw data into HDFS format and divides blocks based on the data size. In the above example the blocks B1to B4 are distributed among the DataNodes.

Step 5: Client "A" sends the three copies of each data block to different DataNodes.

Step 6: In this step, client "A" sends a MapReduce job (job1) to the JobTracker daemon with the source data file name(s).

Step 7: JobTracker sends the tasks to all TaskTrackers holding the blocks of the data.

Step 8: Each TaskTracker executes a specific task on each block and sends the results back to the JobTracker.

Step 9: JobTracker sends the final result to Client "A". If client "A" has another job that requires the same datasets it repeats the set 6-8.

Step10: In native Hadoop client "B" with a new MapReduce job (job2) will go through step 1-5 even if the datasets are already available in HDFS. However, if client "B" knows that the data exists in HDFS, it will send job2 directly to JobTracker.

Step 11: JobTracker sends job2 to all TaskTrackers.

Step12: TaskTrackers execute the tasks and send the results back to the JobTracker.

Step 13: JobTracker sends the final result to Client "B".

We can see that there is independency between jobs because there are no conditions that test the relationship between jobs in Native Hadoop. So, every job deals with the same data every time it gets processed. In addition, if we have the same job executed more than one time; it reads all the data every time, which can cause weakness in Hadoop performance.

## V. PROPOSED SOLUTION

In current Native Hadoop architecture, NameNode knows the location of data blocks in HDFS. NameNode is also responsible for assigning jobs to the clients. Knowing which DataNode contains these blocks, with the required data. NameNode should be able to direct the jobs to read the specific DataNodes without going through all DataNodes. And Etcd3 is useful for finding the sequence of the data or the process. The Etcd3 is used to hold a sliding window to keep old events so that disconnecting will not cause all events to be lost. So when we

use native hadoop along with the Etcd3 in the biomedical industries where using large collection of the database will be benefited by this technique. Because this combined method will overcome the disadvantages like sequence access of data and mutation of the data from the data. This method will help in efficient retrieval of sequence of data from the database and keep track of the old tracking information.

## VI.CONCLUSION

Future works can be carried out over the Native hadoop and the Etcd3 to minimize the network disruption. Because the network disruption will abort the operation performed inside the system and sometimes may result in aborting the sequence of the information in the database. There is loss in the security of the data and information stored in the cloud. So in future work it must be done how to enhance the security feature and the network disruption.

## VII. ACKNOWLEDGEMENT

I am thankful to Prof. K. Sivakumar and Dr.G.Rajiv Suresh Kumar for them guidance and support to pursue this work.

## REFERENCES

[1] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, "A comparison of approaches to large-scale data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 165-178

[2] Hadoop, "Hadoop: http://hadoop.apache.org/zookeeper/, accessed on 15 June 2010," 2010.

[3] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears, "Online aggregation and continuous query support in mapreduce," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 1115-1118.

[4] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce Online," in *NSDI*, p. 20.

[5] S. Leo, F. Santoni, and G. Zanetti, "Biodoop: Bioinformatics on Hadoop, Parallel Processing Workshops, International Conference on, pp. 415-422, 2009 International Conference on Parallel Processing Workshops, 2009," 2009.

[6] Y. Liu, B. Schmidt, and D. L. Maskell, "DecGPU: distributed error correction on massively parallel graphics processing units using CUDA and MPI," BMC bioinformatics, vol. 12, p. 85.

[7] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S.

Gabriel, and M. Daly, "The Genome Analysis Toolkit: a MapReduce framework for analyzing nextgeneration

DNA sequencing data," Genome research, vol. 20, pp. 1297-1303.

[8] P. Khatri and S. DrÄƒghici, "Ontological analysis of gene expression data: current tools, limitations, and open

problems," Bioinformatics, vol. 21, pp. 35873595, 2005.

[9] H. Mathkour and M. Ahmad, "Genome Sequence Analysis: A Survey," Journal of Computer Science, vol. 5,

2009.

[10] A. Matsunaga, M. Tsugawa, and J. Fortes, "CloudBLAST: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications," in eScience, 2008. eScience '08. IEEE Fourth International Conference on, 2008, pp. 222-229.

[11] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for MapReduce in a

cloud," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, p. 58.