Hadoop Based Enhanced Cloud Architecture For Bioinformatic Algorithms

Hamoud Alshammari[1], Hassan Bajwa[2] and Jeongkyu Lee[1]
Department of Computer Science [1]
Department of Electrical Engineering[2]
221 University Ave, University of Bridgeport,
Bridgeport, CT, USA

***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 present an enhanced Hadoop architecture to reduce computation by utilizing "Common Features" before performing redundant computation. The enhanced Hadoop architecture allows the jobs to share these common features among them. The common features describe the contents of data in blocks and can be used to determine DataNodes that store the required data.**

***Keywords* —** BigData, Hadoop, Hadoop Architecture, MapReduce.

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. Hadoop based cloud architecture is composed of Hadoop Distributed File System (HDFS), MapReduce programming model and Apache Zookeeper 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]. ZooKeeper is critical component of the infrastructure, it provide coordination and messaging across applications [2]. The ZooKeeper capabilities include naming, distributed synchronization, and group services. Hadoop framework leverage on large-scale data analysis by allocating data-blocks among distributed DataNodes.

Hadoop Distributed File System (Fig 1 and Fig2) allows the distribution of the data set into many machines across the network that can be logically combed for processing.

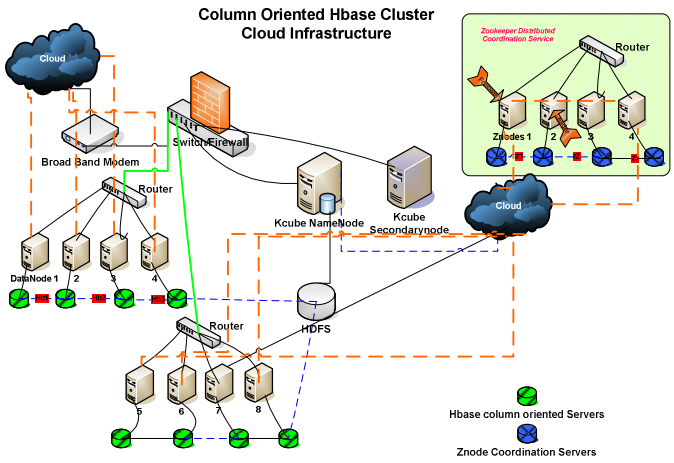

Figure 1: Hadoop Architecture

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 2). HDFS is architected to have the block fault and replication tolerance. NameNode is responsible to maintain a healthy balance between disks 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].

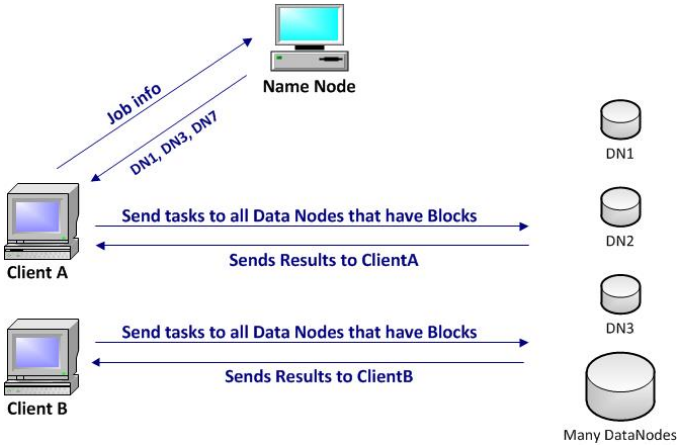

Figure 2: DataNodes and Task Assignment

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. 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: fininding 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 Hadoop MapReduce 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. However, when we execute the MapReduce job in the same cluster for more one time, each time it takes the same amount of time, which is the main problem that we try to represent and propose a solution to improve the related jobs each time that jobs get executed.

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 is looking for Super sequence with sub-sequence already has been searched will have to go through the BigData again. Thus the cost to perform the same job will stay the same each time.

IV. CURRENT ARCHITECTURE

In current Hadoop and MapReduce architecture, the client first sends a job to the cluster administrator,

which is the NameNode or the master of the cluster. Before that, the data source file 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 Data Nodes within the cluster. Any new job 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 result will be stored in also in HDFS.

In current architecture, data follows the concept of write-one read-many, so there is no ability to do any changes in the source file in HDFS (Figure 3). However, each job has to get the data from all blocks that store the source data file in HDFS. Some research groups have already presented solution to address issue of latency while reading data from DataNodes [11]. In current Hadoop/MapReduce architecture (Figure 3) multiple job that need the same data set work independent of each other. We also noticed that searching for the same sequence require same amount of time each time we execute the job, also searching for the subsequence of a sequence that has already been searched require the same amount of time.

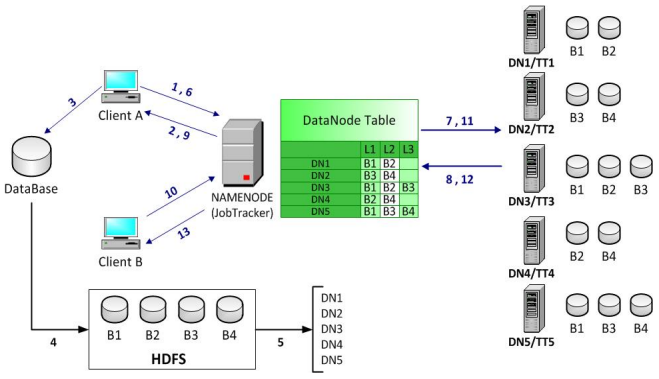

Figure 3: Current Hadoop/MapReduce Architecture

HADOOP MAPREDUCE WORKFLOW:

Workflow of current Hadoop architecture is shown in (Figure 3). The process of sending a MapReduce job within the current architecture of Hadoop goes through the following steps:

a) Client A sends a request to NameNode that includes the need to copy the data files to DataNodes with 3 copies.
b) NameNode replays with the IP address of five DataNodes (DN1, DN2, DN3, DN4, DN5).
c) Client A accesses the shared data files (DB).
d) Client A reformats the data files into HDFS format in many Blocks (B1, B2, B3, B4).
e) Client A sends the blocks to the DataNodes with three copies for each block.
f) Client A sends a MapReduce-job1 to the JobTracker daemon with the name of the data.
g) JobTracker sends the job to all TaskTrackers who hold the blocks of the data.
h) Each TaskTracker executes a specific task on each block and sends the results back to the JobTracker.
i) JobTracker sends the final result to Client A.
j) Client B sends a new MapReduce-job2 to the JobTracker with the same name of the data.
k) JobTracker sends job2 to all TaskTrackers.
l) TaskTrackers execute the tasks and send the results back to the JobTracker.
m) JobTracker sends the final result to Client B.

V. PROPOSED SOLUTION

In current 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.

We propose changes in Hadoop framework process that are between the NameNode and the DataNodes. We focus on having a kind of data in a table that carries information related to the location of the blocks that give specific results for a job. So, for the other job that needs to go through the same data, it should only read the data from these specific blocks on the cluster without going through the whole data again. This framework is supposed to give a better efficiency than the current architecture.

VI. PROPOSED ARCHITECTURE

Proposed Hadoop/MapReduce architecture is same as the original Hadoop in terms of hardware, network, and nodes. However, the software level has been enhanced. We add some features in NameNode that allow it to save some data by adding a look up table named as Common Job Blocks (CJB) Table. This table stores some information about the jobs and the blocks that these jobs got their results from. This architecture allows the related jobs to go and get results from these blocks without checking the entire cluster. CJB table is related to only one HDFS data file, which means that there is only one table for each file in HDFS. Also, it gets updated with the new common features (for example, common sequences in the project of human genome finding sequence alignment) each time a new job reaches this file, so this table gets resized

dynamically and dramatically. Consequently, the size of this table should be controlled and limited to a specific size to keep the architecture reliable and efficient. It consists of three main components or columns (Table 1), which are:

Table 1: Common Job Blocks table components

Common Job Name	Common Feature	Block Name		
Sequence_Alignment	GGGATTTAG	B1	B2	B3
	TTTAGA	B1	B4	
Fining_Sequence	TTTAGCC	B3	B6	
	GCCATTAA	B1	B3	B4
	AATCCAGG	B3	B5	

1) *Common Job Name (CJN)*

This component represents of a shared name of a common job that each MapReduce job must be submitted using this common name to get the benefit of the proposed architecture. This name should be represented to the client when the user creates a MapReduce job program to allow him to select a job name from a list of job names that already identified in advance to be used later on any shared MapReduce job.

That feature plays the main role of the way that NameNode knows this job from its name and matches it with the one that the NameNode has on the CJB table. Since the step of selecting the common job name happens by the user, he has to know in advance the available common job names that he can use. So, on behalf of that, a proposed way is to build a library that contains of a list of per-coded jobs that could be used for common duties. These listed jobs have a description that explains some details that explains how to use it and what are the expectations of using it.

2) *Common Feature*

This feature represents of the shared data entry that the jobs carries when it sent for execution. This feature stays beyond the idea of having a common feature between some jobs. So, this feature is mainly related to a specific job, which means each job has its own feature format. For example, in sequence alignment job, we have a sequence of letters that the job tries to find the identical sequence in the source data file in HDFS. So this job does not take any other type of data that is not on the specified format.

When a sub-common feature (e.g. sub-sequence in human genome example) arrives to the NameNode with a new job, the old common feature will be replaced with the shortest one. However, there must be a limit of its length to not be less than that limit for the job to be acceptable and executable. For the common feature this limited length is related to each job separately.

3) *Block Name*

Which is the location of these common features, which means in which blocks in the cluster they are located. This feature allows the NameNode to direct the job to get data only from the DataNodes that store these blocks on their HDFS. CJB table stores all blocks that are related to the results of the common feature. So, the number of columns/blocks increases and decreases every time the number of blocks gets changed. So, the maximum expected number of columns/blocks in the table is the real number on the all blocks that represent the whole file in HDFS, which means that the results can be found in each block on the HDFS. Consequently, the minimum expected number of columns/blocks in the table is zero, which means there are no results that are related to the common feature for that job in the HDFS.

HOW BIG IS CJB TABLE?

Common Job Blocks table should not exceed to be a huge table that makes the process of research for a common feature takes long time. So, the size can be limited by following one of the algorithms that solves this issue like Leaky Bucket Algorithm.

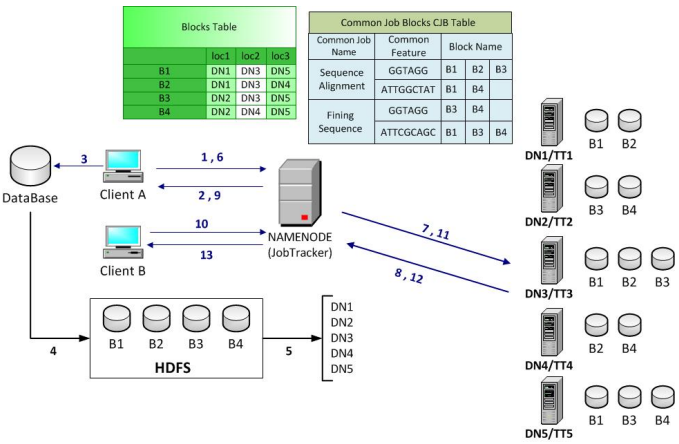

Figure 4: Enhanced Hadoop/MapReduce Architecture

HADOOP BASED ENHANCED CLOUD ARCHITECTURE WORKFLOW

Figure 4 proposes the steps that an enhanced Hadoop based architecture follow to process any related two jobs as following:

a) Client A sends a request to NameNode that includes the needs to copy the data files to DataNodes with 3 copies.
b) NameNode replays with the IP address of five DataNodes (DN1, DN2, DN3, DN4, DN5).
c) Client A accesses the shared data files (DB).
d) Client A reformats the data files into HDFS format in many Blocks (B1, B2, B3, B4).
e) Client A sends the blocks to the DataNodes with three copies for each block.
f) Client A sends a MapReduce-job1 to the JobTracker with the name of the data.
g) JobTracker sends the job to all TaskTrackers who hold the blocks of the data.
h) Each TaskTracker executes a specific task on each block and sends the results back to the JobTracker. In this step, let us assume only three TaskTrackers (TT2, TT4, TT5) found the answer in their blocks.
i) JobTracker sends the final result to Client A. In this step, NameNode keeps the names of the blocks that produced the result in a local lookup table (CJB table) by the Common Job Name (Job1) that has common feature as what we explained earlier.
j) Client B sends a new MapReduce-job2 to the JobTracker with the same common job name and same common feature or super-sequence of it. In this step, Job2 has the same common name of Job1 but with different or same details.
k) JobTracker sends job2 to those TaskTrackers who hold the blocks that have the first result of MapReduce-job1 (DN2, DN4, DN5). In this step, JobTracker starts with checking the CJB table first to find is the new job has the same common name and common features of any previous ones or not. (In this case yes). Then JobTracker sends Job2 only to TT2, TT4 and TT5. We may assume here the lookup table will be updated with more details OR just keep it as it because every time we have a new job that will carry the same name of Job1.
l) TaskTrackers execute the tasks and send the results back to the JobTracker.
m) JobTracker sends the final result to Client B.

VII. Conclusion

We proposed an enhanced Hadoop features of the current Hadoop architecture to decrease the computation time and cost. By having a dynamic dataset, which is a Common Job Blocks table to determine the exact blocks that carries the data that related to the results. Some future work could be implemented to improve some facilities of the new architecture like develop a complete libraries that related to sets of big data types and develop an algorithm that related to the part of the size of the Common Job Blocks table to keep it reliable.

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

[8] P. Khatri and S. DrÃƒghici, "Ontological analysis of gene expression data: current tools, limitations, and open problems," *Bioinformatics,* vol. 21, pp. 3587-3595, 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.