Distributed Gene Clinical Decision Support System Based on Cloud Computing

Bo Xu, Changlong Li, Hang Zhuang, Jiali Wang, Qingfeng Wang, Chao Wang, Xuehai Zhou
School of Computer Science and Technology
University of Science and Technology of China
Hefei, China
{xubo245, liclong, zhuangh, ustcwjl, qfwangyy}@mail.ustc.edu.cn, {cswang, xhzhou}@ustc.edu.cn

Abstract—The clinical decision support system can effectively solve the limitations of doctors' knowledge, reduce misdiagnosis and help enhance health. The traditional genetic data storage and analysis technology based on the stand-alone environment have limited scalability, which has been difficult to meet the computational requirements of rapid genetic data growth. In this paper, we propose a distributed gene clinical decision support system, which is named as GCDSS. We implemented a prototype based on cloud computing. To speed up the data processing of GCDSS, we present a novel distributed read mapping algorithm CloudBWA that leverages batch processing strategy to map reads on Apache Spark. Evaluations show that GCDSS and its component CloudBWA achieve outstanding performance and excellent scalability. Compared with distributed algorithms, CloudBWA achieves up to 2.63 times speedup over SparkBWA.

Keywords—Gene Clinical Decision Support System, Cloud computing, Spark, Alluxio, Genetic data analysis, Read mapping

I. INTRODUCTION

Clinical decision support system (CDSS) provides clinicians, staff, patients, and other individuals with knowledge and person-specific information to enhance health and health care. CDSS can effectively solve the limitations of doctors' knowledge, reduce misdiagnosis and relatively lower the cost of health care, so as to guarantee the quality of medical care. Genetic diagnosis can achieve the goal of early detection, early discovery, early prevention, and early treatment.

With the development of next-generation sequencing (NGS) technology, the number of newly sequenced data has exhibited an exponential increase in recent years. It has become a huge challenge that how to store and analyze the big data of gene. So faster genetic data storage and analysis technologies are needed to keep pace. Moreover, time is life in the clinical field, especially the emergency. Therefore, it is of great significance to accelerate the processing and analysis of genetic data for CDSS. However, the traditional genetic data storage and analysis technology based on the stand-alone environment have limited scalability, which has been difficult to meet the computational requirements of rapid data growth.

To solve this problem, we propose GCDSS, a distributed gene clinical decision support system based on cloud computing. There are two important challenges in the process of implementing GCDSS and improving its performance.

The first challenge is how to design and implement a distributed genetic data analysis pipeline framework. Due to genetic data analysis involves a large amount of data, varied data formats and many complicate analysis steps and other factors, it has become a great challenge that how to design and implement a unified distributed genetic data analysis pipeline framework.

The second challenge is how to solve the problem that the scalability of traditional read mapping algorithms is limited. Read mapping is the first step and also a very time-consuming step in the whole genetic data analysis pipeline. A sample typically produces billions of reads. It is critical and quite difficult for subsequent analysis that quickly and accurately mapping these reads to the reference genome.

To tackle these challenges, we take distributed storage, distributed computing framework, and distributed algorithms into consideration, and exploits cloud computing technology to parallelize genetic data analysis pipeline. We claim following contributions and highlights:

1) This paper designs a distributed genetic data analysis pipeline framework in GCDSS and implements its prototype based on cloud computing. GCDSS designs unified pipeline that effectively integrates read mapping and calibration, the variant discovery and genotyping, disease identification and analysis into the framework.

2) This paper presents a novel distributed read mapping algorithm CloudBWA. It leverages Apache Spark [1] to enable traditional BWA-MEM [2] algorithms run in a horizontally scalable distributed environment. To improve the performance of read mapping algorithm, we design and implement batch processing strategy.

The rest of this paper is organized as follows. Section II provides the background and related work. In Section III, we describe the design of GCDSS. Section IV shows the design of CloudBWA. Section V shows our experiments and evaluation results. In Section VI, we summarize the conclusions and discuss future research work.

II. BACKGROUND AND RELATED WORK

A. Genetic Data Analysis Pipeline

Over the past few years, there have been tremendous efforts to implement the distributed framework. AMPLab of UC Berkeley [3, 4], UCLA [5], Broad Institute of MIT and Harvard [6], Illumina [7] and other institutions are developing distributed genetic data analysis pipeline framework. AMPLab has presented distributed genomics data formats Adam [4] and distributed variant discovery and genotyping algorithms Avocado [3], but it has not implemented distributed read mapping. UCLA has presented CS-BWAME [5] for distributed read mapping, and implemented distributed sort and mark duplicates (MD), but base quality score recalibration
(BQSR) and Indel realignment are in progress. We have done related efforts and research works in previous [8-10].

B. Read Mapping

BWA [2, 11, 12] is the best popular read mapping tool at present. BWA includes BWA-backtrack [11], BWA-SW [12] and BWA-MEM [2]. But its scalability is limited. CS-BWAMEM is an ultrafast and highly scalable read mapping aligner built on top of cloud infrastructures [5]. At present, CS-BWAMEM only supports paired-end read mapping. SparkBWA is a new tool that integrates the BWA[2, 11, 12] into the Spark framework [13]. However, SparkBWA will be wrong when numPartitions size is larger than the number of Spark work in the cluster. Limited numPartitions size may result in uneven distribution of data and calculations, which ultimately results in performance loss. The I/O overhead of SparkBWA is very large because it needs to read and write disk several times.

III. DESIGN OVERVIEW

This section overviews the design of GCDSS, and describes its workflow, implementation, and API respectively. This section focuses on the first challenge.

A. System Workflow

The difference between GCDSS and traditional CDSS is that GCDSS mainly uses the genetic data to analyze and process, rather than mainly uses the data of traditional Chinese medicine or modern medicine to build the system.

As shown in Figure 1, the system workflow of GCDSS consists of three phases: NGS data processing, variant discovery and genotyping, disease identification and discovery.

1) NGS data processing

The first phase is NGS data processing, which mainly consists of read mapping and calibration.

After obtaining sequencing data from sequencer, we need map billions of raw reads to reference genome and obtain the most probable location of every read, which is usually called as read mapping. Read mapping is quite difficult and critical for subsequent analysis that quickly and accurately mapping billions of reads to the reference genome. To address the scalability problem of traditional read mapping algorithms, we present a distributed read mapping algorithm CloudBWA. We will describe its detail in Section IV.

Calibration consists of MD, local realignment (LR) and BQSR. The data that are returned by read mapping are unordered.

To reduce the overhead of computation in a distributed environment, we need sort all alignment reads in the resilient distributed dataset (RDD) [1] before calibration. We sort the alignment reads by the name and location of the reference sequence that matches the read.

2) Variant discovery and genotyping

The second phase is variant discovery and genotyping, which mainly discover possible variant in analysis-ready reads, and then genotyping. Because the majority of variants is single nucleotide polymorphism (SNP), this paper mainly focuses on the variant discovery and genotyping of SNP, and considers insertion-deletion polymorphism (Indel).

Variant discovery mainly traverses every mapped read in RDD, and finds all the variations in each read, and then performs the Map and ReduceByKey to count the number of each variation and sorts them, and finally returns the processed variation information in the form of variant RDD.

Genotyping uses biallelic genotyper [3] to obtain the genotype of the sample. It first scores variants in variant RDD and gets observations, and then turns single variant observation into genotype call, and finally creates a genotype RDD.

We need to post processing after genotyping, including filtering genotypes with low coverage, genotype refinement, filtering reasonable SNPs and Indels by comparing with known database and so on.

3) Disease identification and discovery

The third phase is disease identification and discovery. As shown in Figure 2, this phase consists of two steps: the construction of the associated database and association analysis.

The construction of the associated database mainly uses the public database to build an associated database of the discovered variations and known diseases. Each variation in variation database includes the chromosome name, position, variant id, reference base, alternate base and so on. Each record in mapping database of variation and disease includes id, the locus id the SNP is on, locus symbol and SNP id and so on. Disease database describes a variety of disease-related clinical features, pathogenesis, diagnosis methods, treatment methods, latest research results and so on.

The main steps to build an association database are:

a) Preprocessing variation database, disease database and mapping database of variation and disease.

b) Analyzing comprehensively variation database and mapping database of variation and disease, filtering out the variations that are not directly related to disease.

c) Obtaining association database by integrating processed data with disease database. It should include variation and corresponding disease information.

Association database includes the disease id, chromosome name, position, variant id, reference base, alternate base, locus symbol, title, method and link of corresponding disease description on the disease website. The link can be linked to the latest disease website page, which contains the latest information.

Association analysis is designed to obtain possible disease information by comparing variations of sample with association database. The main steps of association analysis are:
a) Preprocessing variations of the sample and association database, including separating variations, extracting data, unifying the chromosome name, etc.
b) Generating key by combining chromosome name, position, reference base, alternate base, and then join variant RDD into association database RDD by invoking Spark join function.
c) Post processing data, including converting data, generating a diagnostic report of the sample and storing data into distributed file system or database.

In order to facilitate distributed genetic data analysis, we implement different distributed algorithms, including base algorithm library (BAlib), extract-transform-load library (ETLlib), upload and download library (UDlib) and conversion library (Clib). BAlib is a base algorithm library for distributed genetic data processing and analysis. ETLlib is a library for the extraction, transformation, and loading of genetic data. UDlib is a library for converting different data formats.

C. System API

In order to facilitate the user to use GCDSS, we provide application programming interfaces (APIs). In accordance with the previous system workflow and implementation, GCDSS provides the corresponding APIs, including upload and download operations APIs, ETL operations APIs, conversion operations APIs and different algorithms APIs.

IV. DISTRIBUTED READ MAPPING ALGORITHM

In order to solve the scalability problems of traditional read mapping algorithms, this paper presents a distributed read mapping algorithm based on cloud computing, which is called as CloudBWA. This section describes its framework, workflow, and API respectively.

A. Algorithm framework

As shown in Figure 3, CloudBWA employs usually Master-slave framework. Master is primarily responsible for manage metadata and cluster, which combines Spark Master, Alluxio Master, and HDFS NameNode. Slave consists of two layers: storage layer and compute layer. The first layer is the storage layer. In order to speed up read and write, we employ a memory-based distributed file system (DFS) Alluxio as primary storage component instead of the traditional disk-based DFS. HDFS is only used as persistence in the storage layer. The second layer is the compute layer. It is based on Apache Spark. We employ Adam and SAM tools to transform different genetic data formats, and employ BWA tools to read mapping in each node. Spark cannot directly invoke the read mapping algorithm of BWA tools, which are written in C language. In order to integrate read mapping algorithms into Spark, we employ jBWA to read mapping because jBWA implements the function of Java invoking BWA by using Java Native Interface (JNI) in single node [17].
B. Algorithm Workflow

CloudBWA mainly combines Spark and BWA tool to distributed read mapping. As shown in Figure 4, the CloudBWA algorithm consists of three phases: the data preprocessing phase, the Map phase, and the post-processing phase.

1) Preprocessing

The preprocessing phase mainly reads data from the DFS and preprocesses, mainly including loading data, converting format, filtering data, pairing reads, and caching data in memory.

2) Map

This phase employs batch processing technology to speed up data processing. As shown in Figure 4, a new RDD of reads will be generated after preprocessing. In order to facilitate the description, each node in Figure 4 is assumed to have only one partition. Assuming that the data distribution is sufficiently uniform, the number of reads per partition is assumed to be m pairs. Paired-end reads of Partition1 are named as read1,1 to read1,m. The paired-end reads in different partitions are generally different. CloudBWA provides two output modes: SAM and Adam mode. Assuming the size of the batch is k. The main steps of the map phase are:

a) Loading reference genome. The jBWA is invoked to load the reference genome in mapPartitions function.

b) Read mapping. When the size of the batch reaches k, it will invoke jBWA to read mapping. If the remaining data of the partition is less than k, they will be mapped.

c) Unifying the expression. Since jBWA may return multiple results, they need to be filtered and transformed.

d) Adam mode processing. If the output format is Adam, CloudBWA will convert result into Adam format.

e) After processing a batch, the next batch will be processed from b) until the genetic data of the partition is processed.

f) When a partition is processed, CloudBWA will release related data and operations of jBWA. When all the partition is processed, CloudBWA starts the next phase.

3) Post processing

CloudBWA needs to further process after map phase. The main steps include obtaining information and generating RDD, then saving the RDD into DFS or returning the RDD.

C. Algorithm API

In order to facilitate the user to better use CloudBWA, we also provide APIs. The APIs includes two different output modes that are SAM and Adam, different input formats that are FASTQ and Adam, different DFSs that are HDFS and Alluxio. CloudBWA also provides APIs for converting a single SAM format string into Adam format data and converting SAM format RDD into Adam format RDD.

V. EVALUATION

In this section, we focused on the evaluation of CloudBWA. We also verify the feasibility of the GCDSS prototype.

All our experiments were performed on a cluster with 8 nodes. The operation system of each node is Ubuntu-14.04.1. Each node has a dual-core Intel Xeon W3505 CPU with 22GB of RAM. The version of Apache Spark is 1.5.2. The Alluxio version is 1.3.0. The version of HDFS is 2.6.0. In order to debug and analysis algorithms, we employ wgsim [18] to generate simulation data. The version of wgsim is 0.3.2. The parameters of wgsim are set as default besides description.

A. Evaluation of CloudBWA

CloudBWA is evaluated from different aspects, including the impact of different parameters on CloudBWA, the scalability evaluation, the performance comparison and so on.

1) Performance evaluation

a) Impact evaluation of batch size and output mode

The experimental data are generated by wgsim, which have 20 millions of reads and its length is 50 base. The memory size of executor in Spark is set as 20G. The size of numPartitions is 32. Adam uses GZIP to compress. Reference is chromosome 1 of GRCh38. DFS is HDFS. The input format of reads is FASTQ.

Figure 5 shows impact evaluation of batch size and output mode on CloudBWA. The experimental result shows that SAM mode is faster than Adam mode because Adam format needs more computation. When batch size is 1 reads, the runtime of SAM and Adam mode is long. When increasing the batch size, the runtime of SAM mode will fall first and then rise, and finally become stable. Adam mode will rise when batch size is large.

b) Impact evaluation of different input data format and numPartitions size

The experimental data are the same as the first experiment. The output mode is SAM mode. The size of the batch is 10 reads.
The DFS is Alluxio. CloudBWA uses different numPartitions size. Other environments are the same as the first experiment.

Figure 6 shows impact evaluation of different input data format on CloudBWA. The result shows that Adam format achieves average 9.6% performance improvement over FASTQ format because FASTQ format needs extra transformation overhead. With the increase in numPartitions, the runtime of CloudBWA increase quickly and then increase slowly.

Figure 7 shows the speedup improvement of CloudBWA by increasing the number of nodes. The experimental result shows that CloudBWA achieves near-linear speedup when increasing the number of nodes from 1 to 8 in the cluster.

We compare CloudBWA with SparkBWA and CS-BWAMEM, which are one of the state-of-the-art read mapping algorithms in the distributed environment. The version of CS-BWAMEM is 0.2.2, its output format is Adam and batch size is 100 reads. The version of SparkBWA is 0.2, its output format is SAM and numPartitions size is 8, but SparkBWA uses two threads in each node. The version of CloudBWA is 1.0.1, its numPartitions size is 16 and batch size is 10 reads. The three algorithms use 8 nodes and 16 cores in the cluster.

Figure 8 shows performance comparison with distributed algorithms. Figure 8(a) shows CloudBWA is faster than SparkBWA and CS-BWAMEM in a different number of reads. The Adam mode of CloudBWA achieves average 1.84 times speedup over CS-BWAMEM. The SAM mode of CloudBWA achieves up to 2.63 times speedup over SparkBWA. Figure 8(b) shows CloudBWA is faster than SparkBWA and CS-BWAMEM in different length of reads. The Adam mode of CloudBWA achieves up to 2.22 times speedup over CS-BWAMEM. The SAM mode of CloudBWA achieves average 1.44 times speedup over SparkBWA.

B. Evaluation of other GCDSS’s components
CloudBWA is kernel algorithm in GCDSS. Besides, we also evaluate other GCDSS’s components. We focus on performance and scalability evaluation.

1) Calibration evaluation
The experimental data are the same as the experiment of A 3) in this section. The length of reads is 50.

a) Performance evaluation of calibration
Figure 9 shows performance evaluation of calibration. The experimental result shows that sort is the fastest and LR is the slowest in calibration. MD is slower than BQSR at the beginning, but MD is faster when the number of reads is larger than 20 million. With the increase in the number of reads, the four operations’ runtime of calibration is also near-linear increasing.

b) Scalability evaluation of calibration
Figure 10 shows scalability evaluation of calibration. The experimental result shows that the four operations of calibration achieve near-linear speedup when increasing the number of nodes from 1 to 8 in the cluster.

2) Variant discovery and genotyping evaluation
The experimental data are the same as the experiment of A 3) in this section. The length of reads is 50.

a) Performance evaluation
Figure 11 shows performance evaluation of variant discovery and genotyping. The experimental result shows that with the increase in the number of reads, the runtime of variant discovery and genotyping both are near-linear increasing.
In this paper, we present GCDSS, a distributed gene clinical decision support system based on cloud computing. We design a unified distributed genetic data analysis pipeline framework in GCDSS. To speed up the data processing of GCDSS, we present a novel distributed read mapping algorithm CloudBWA that leverages batch processing strategy to map reads on Spark. The evaluations show that CloudBWA achieves outstanding performance and excellent scalability. It achieves up to 2.63 times speedup over SparkBWA. Besides, we also evaluate other GCDSS’s components, including calibration evaluation, variant discovery and genotyping evaluation, disease identification and discovery evaluation. The results show that GCDSS achieves outstanding performance and excellent scalability.

In the future, we plan to explore different technologies to improve performance and increase the type of disease analysis.

**ACKNOWLEDGMENT**

This work was supported by the National Science Foundation of China (No. 61379040), Anhui Provincial Natural Science Foundation (No.1608085QF12), Suzhou Research Foundation (No.SYG201625), Youth Innovation Promotion Association CAS (No. 2017497), and Fundamental Research Funds for the Central Universities (WK2150110003). Chao Wang is the corresponding author.

**AVAILABILITY**

An open source GCDSS (GNU GPL v.2) is freely available at https://github.com/xubo245/GCDSS.

**REFERENCES**

[1] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," HotCloud, vol. 10, pp. 10-10, 2010.

[2] C. Wang, X. Li, X. Zhou, Y. Chen, and R. C. Cheung, "Big data genome sequencing on zynq based clusters," in Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, 2015, pp. 631-646: ACM.

[3] Y.-T. Chen et al., "CS-BWAMEM: A fast and scalable read aligner at the cloud scale for whole genome sequencing," in High Throughput Sequencing Algorithms and Applications (HITSEQ), 2015.

[4] G. A. Auwera et al., "From FastQ data to high - confidence variant calls: the genome analysis toolkit best practices pipeline," Current Protocols in Bioinformatics, pp. 11.10. 1-11.10. 33, 2013.

[5] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[6] B. Xu et al., "Distributed genome sequencing pipeline on Hadoop," in Proceedings of the 17th IEEE/ACM International Symposium on Cloud, Internet and Grid Computing, 2017, pp. 758-761: IEEE Press.

[7] B. Xu, C. Li, H. Zhuang, J. Wang, Q. Wang, and X. Zhou, "Efficient Distributed Smith-Waterman Algorithm Based on Apache Spark," in Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on, 2017, pp. 608-615: IEEE.

[8] G. A. Auwera et al., "From FastQ data to high - confidence variant calls: the genome analysis toolkit best practices pipeline," Current Protocols in Bioinformatics, pp. 11.10. 1-11.10. 33, 2013.

[9] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[10] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[11] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[12] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[13] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[14] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[15] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[16] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[17] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.

[18] M. Parsian, Data Algorithms: Recipes for Scaling Up with Hadoop and Spark." O'Reilly Media, Inc., 2015.