Gene Sequence Input Formatting and MapReduce Computing

Xiaolong Feng, Jing Gao*

College of Computer and Information Engineering
Inner Mongolia Agricultural University
Hohhot 010018, China
E-mails: fengxl@imau.edu.cn, gaojing@imau.edu.cn

**Corresponding author*

Received: October 30, 2018 **Accepted: April 25, 2019**

Published: June 30, 2019

Abstract: *Considering the limitations of the application programming interface (API) of Hadoop in gene sequence computing, this paper puts forward an input formatting method that reads the format of gene sequence as key-value pairs in the form of records. This method relies on the rewriting of Hadoop source code, which is an extension of platform function, and eliminates the need to preprocess data with other tools. On this basis, a MapReduce computing model was designed for distributed parallel computing of gene sequence alignment tasks. Experimental verification shows that the proposed method can read many kinds of gene sequence files effectively on Hadoop, and the proposed model can realize distributed parallel computing of gene sequence alignment. The research findings provide a valuable reference for bioinformatics computing tasks on Hadoop platform.*

Keywords: *Input formatting, MapReduce, Gene sequence, Sequence alignment, Short reads mapping.*

Introduction

Gene sequence computing mainly deals with big data. The large data scale poses a huge computing load, requiring a long time to process. This calls for the improvement of big data technologies [7]. One of the popular big data processing frameworks is called Hadoop, which allows data to be partitioned and computed separately on each node in the cluster. Thus, this distributed computing platform is applicable to many gene computing tasks [14]. However, Hadoop only offers a general solution to big data problems, failing to meet the specific needs of gene sequence computing. In fact, there is not yet any stable, efficient and scalable distributed parallel computing model for gene sequence processing.

To make up for the gap, this paper designs a suitable gene sequence input formatting method and a MapReduce computing model based on Hadoop platform, aiming to facilitate the development of bioinformatics computing applications with an integrated research tool. The proposed solution gives full play to the advantages of Hadoop in distributed parallel computing and reduces the time consumption of bioinformatics computation.

Research background

Hadoop is a popular distributed framework for big data processing, which is highly reliable, scalable, efficient, fault-tolerant and cost-effective [4]. The framework mainly consists of the Hadoop distributed file system (HDFS), Hadoop YARN and Hadoop MapReduce. Specifically, the HDFS is a distributed file system that stores files with data blocks in distributed cluster, and ensures the data validity with a good fault-tolerance mechanism. The YARN is a resource scheduling system that properly allocates CPU, memory and IO resources from the cluster to

different tasks, and ensures the resource exclusiveness in task execution. The MapReduce is a two-phase distributed computing system (the Map phase maps data from one key-value pair to another, and the Reduce phase aggregates data with the same key) and a programming model for distributed systems.

Compared with the message passing interface (MPI), Hadoop can complete each task locally on storage nodes rather than share memory in the storage area network (SAN), eliminating the need to consider the underlying data flow, fault tolerance and parallelism. In addition, Hadoop cluster is built in the data center connected by high-speed network, with the same architecture and management platform, while grid computing runs on heterogeneous platforms with varied network bandwidth. The above comparisons show that Hadoop is a desirable tool for bioinformatics computing, in light of the data scale of gene sequence, the features of the task and the cost of implementation.

The information of genetic sequence often consists of multi-line characters with a fixed format. The commonly used storage formats for nucleotide and amino acid sequences are FASTQ and FASTA, both of which use ASCII characters to represent biological information. In gene sequence computing, the basic unit of data processing involves multiple lines. However, in the application programming interface (API) of Hadoop, the input from files is usually read by single line, i.e. the basic unit of data processing is a single line. It is necessary to make data processing tools both recognizable by Hadoop and satisfy the distributed parallel computing of gene sequence processing. Since the processing data need to be further processed in MapReduce, the format processing method should also be optimized to reduce the complexity and time consumption of MapReduce algorithm.

Literature review

Several big data techniques have been applied to gene sequence computing, such as Big Data-Based Burrows-Wheeler Aligner (BigBWA) [1], Halvade [5] and Seal [13].

BigBWA is the latest parallel aligner of gene sequence, with better performance and scalability than other gene aligners. Compared with the traditional aligners, BigBWA can execute gene sequence alignment in parallel with original source code of aligner, and significantly enhance the performance of gene sequence alignment algorithm. Nevertheless, the data must be formatted (e.g., pair-end sequence merging and tag insertion) externally with Python tools before task submission.

Halvade is a gene sequence alignment framework based on Hadoop 2.0 (java). In the Map phase, the data are split into several parts, and the alignment algorithm is invoked for sequence alignment; in the reduce phase, multiple processing programs are called to complete other tasks of gene sequence analysis. Nonetheless, the multi-threaded approach neither supports distributed expansion nor conforms to the computing platform, because the data input and distribution in gene sequence alignment are executed by a platform-independent data distribution program called Halvade Uploader.

Seal is a top-layer application based on Hadoop platform. This tool, developed with Python, mainly implements MapReduce and HDFS operations. The data also need to be formatted by Python before input.

All the above tools rely on Hadoop's ability of distributed parallel computing to parallelize computing tasks, and outshine serial programs in the performance of gene computing algorithm.

However, there are some details in the workflow that are unsuitable or inconvenient to be solved by the Hadoop API. Some scholars have resorted to third-party programs or independent applications, but sacrificed the model uniformly and computing efficiency.

To solve the problem, some scholars have changed the original format of gene sequence files, making the gene sequence easily to identify in the computing model and suitable for processing with Hadoop API. Taking the BigBWA for instance, the markers < seq >...sequence content... </ seq > were added at the start and the end of a single-ended multi-line sequence, and the markers < part >...sequence content... </ part > were added to a pair-end sequence. After tagging, the sequence content in the computing model was easier to identify, and the model overhead was increased. In data interpretation, the recognized sequence markers need to be removed before reaching the alignment algorithm. In general, the data were preprocessed with tools like Python, shell program and standalone applications, which are independent of Hadoop platform.

Inspired by the previous research, this paper puts forward a gene sequence input formatting method based on rewriting Hadoop source code. The rewriting enables Hadoop to adapt to the processing of gene sequences. By this method, the data are inputted into computing models without the aid of third-party tools. Next, a new MapReduce computing model was designed for the distributed parallel computing of gene sequence alignment. The proposed strategy eliminates data preprocessing through input formatting and reduces the overhead of computing model in data format processing, thus enhancing the efficiency of parallel computing.

Materials and methods

Problem description

Taking FASTQ format (Table 1) as an example, the gene sequence file is a fixed-structure text file containing ASCII characters [3]. Each gene sequence generally consists of four lines: the first line is the sequence ID and basic information, starting with "@"; the second line is sequence content; the third line starts with "+", followed by sequence ID and description information; the fourth line is the quality of the sequence content in the second line.

Table 1. A typical FASTQ file

@ERR000589.1.1 EAS139_45:5:1:1:691 length=51
NAGTTTTTATACGAAGATGTTTCCTTTTCTACCTTTGGTCTCAAAGCGATT
+ERR000589.1.1 EAS139_45:5:1:1:691 length=51
!IIIIIIII?IIG7;I0>69@:0@/8.DA?*0/$682++6I4*++8+0(2/
@ERR000589.2.1 EAS139_45:5:1:1:1580 length=51
NAGATTTTTTTCCCACTCTGTGGGTTGTCTGTTTACTCTGCTGACTGTTAC
+ERR000589.2.1 EAS139_45:5:1:1:1580 length=51
!IIIII7I@7IIIIIICIII:II5-H;.2@4+@101?7-;63+&*62+0#.

In Hadoop, data flows in the form of key-value pairs from files to the Map phase and then to the reduce phase. The FASTQ file can be read by the line-by-line method in Hadoop API (Table 2), with the offset of the line as the key and the content of the line as the value.

After the line-by-line reading, the data are subjected to shuffling, sorting and partitioning. However, the information of the same gene sequence will be separated from each other, making it impossible to complete the subsequent operations. To solve the problem, all information of a

gene sequence should be treated as the basic unit of data operation, that is, to read data from a file in the form of gene record. For the input data in Table 1, the target format is to take the offset of the first line of a gene sequence as the key, and all the contents of a gene sequence as the value (Table 3).

Table 2. Line-by-line reading of the FASTQ file

Key	Value
0	@ERR000589.1.1 EAS139_45:5:1:1:691 length=51
45	NAGTTTTTATACGAAGATGTTTCCTTTTCTACCTTTGGTCTCAAAGCGATT
97	+ERR000589.1.1 EAS139_45:5:1:1:691 length=51
142	!IIIIIIII?IIG7;I0>69@:0@/8.DA?*0/$682++6I4*++8+0(2/

Table 3. Target format for gene sequence input in MapReduce model

Key	Value
0	ERR000589.1.1 EAS139_45:5:1:1:691 length=51 NAGTTTTTATACGAAGATGTTTCCTTTTCTACCTTTGGTCTCAAAGCGATT +ERR000589.1.1 EAS139_45:5:1:1:691 length=51 !IIIIIIII?IIG7;I0>69@:0@/8.DA?*0/$682++6I4*++8+0(2/
194	@ERR000589.2.1 EAS139_45:5:1:1:1580 length=51 NAGATTTTTTTCCCACTCTGTGGGTTGTCTGTTTACTCTGCTGACTGTTAC +ERR000589.2.1 EAS139_45:5:1:1:1580 length=51 !IIIII7I@7IIIIIICIII:II5-H;.2@4+@101?7-;63+&*62+0#.

To realize the target formation, it is necessary to rewrite Hadoop's source code and design new data input algorithms and classes, which satisfy the special requirements of gene record. Considering the numerous file formats in gene sequence computing, the main difficulty lies in making the data input method compatible with as many file formats as possible, that is, creating a suitable data input algorithm. Once the file enters the gene record, the distributed parallel computing of gene sequence can be performed on the MapReduce computing model.

In this paper, the computation task of gene short reads sequence mapping is cited as an example. The short reads mapping, as a common gene sequence alignment, mainly compares various short sequences with reference genome, aiming to disclose the relationship between short sequence and reference genome [10].

To fulfill the task, the first step of MapReduce computing model is to split many gene records into data blocks and allocate them to the multiple nodes of distributed cluster. Then, each node maps the short reads sequence to the reference sequence separately, forming its own result file. Finally, the result files are aggregated into a single result file.

Owing to the difference in gene data sequencing, the input data may be either single-end sequence or pair-end sequence [11]. In the former case, a single data file should be distributed to each node. In the latter case, two data files should be allocated to each node. These two cases reflect the requirement of alignment algorithm [2]. Through formatting of the input data, gene

sequences will not be segmented in data operation, eliminating the abnormalities induced by inconsistent data formats in distributed computing.

Input formatting method

In the recent version of Hadoop 2.7.7 [9], the *InputFormat* interface provides several methods for input data segmentation, which divide the data into splits for a Map task, and then complete the Map processing of all data. These methods include *FileInputFormat* class for file input, *DBInputFormat* class for database input, etc.

The *FileInputFormat* class contains *TextInputFormat* for text file, *SequenceFileInputFormat* for binary file, *FiexedLengthInputFormat* for fixed-length file, *KeyValueTextInputFormat* for the file of key-value pair format, *NLineInputFormat* for the file with specified number of lines per split, and *CombineFileInputFormat* for the file merged from small files [6].

The input format of *TextInputFormat* class is the closest to the target format of gene sequence input, while the other classes are not suitable for gene sequence. The gene sequence file has the following features: the file is a text file containing ASCII characters, the number of characters is not fixed per line, no key and value can be identified in the file content, and the file consists of multiple gene sequences, each of which encompasses multiple lines and should be treated as an inseparable unit. The inheritance relationship of the *TextInputFormat* class and its related classes in Hadoop is shown in Fig. 1, which only lists the attributes and methods related to the design objectives.

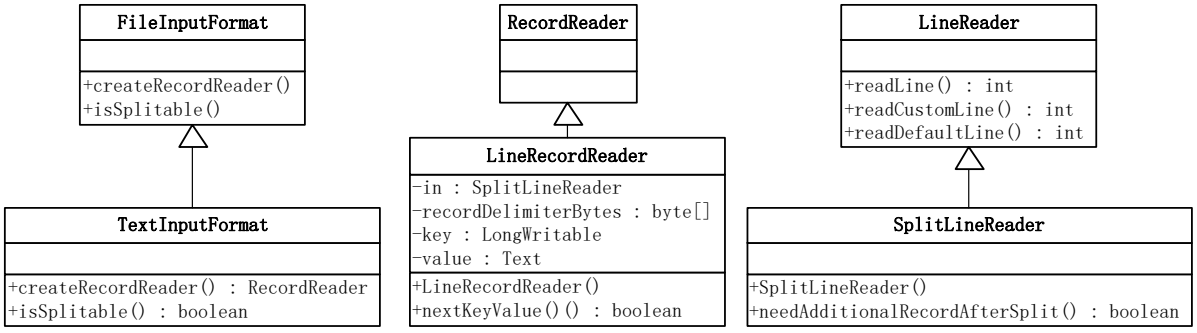

Fig. 1 Inheritance diagram of *textInputFormat* and its related classes in Hadoop 2.7.7

The *TextInputFormat* class in Hadoop source code inherits from the *FileInputFormat* class and overrides the *createRecordReader*() method in the super class for record creation, which returns the object of the *LineRecordReader* class. The *LineRecordReader* class is a subclass of *RecordReader*, which has a data member of *SplitLineReader* type named "in". This data member can invoke *readLine*() method to read text files line by line. The *readLine*() method is not overridden in the *SplitLineReader* class, but inherits from its super class, *LineReader*. Following the principle of *TextInputFormat* class, a series of new classes can be created for the input formatting of gene sequences, and the line-by-line reading of text files can be changed into multi-line reading. As shown in Fig. 2, the new classes can improve the original functions without being separated from the Hadoop environment. In this way, the new classes are compatible with the new data input format, without affecting the subsequent MapReduce operations.

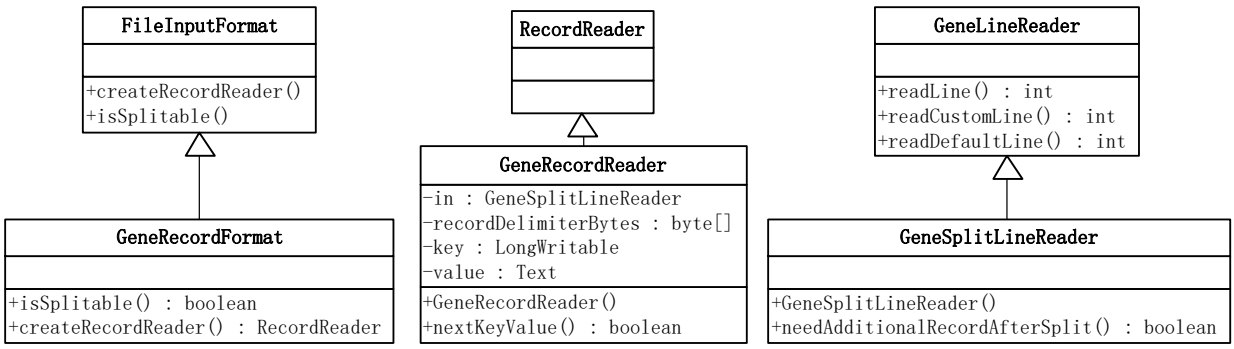

Fig. 2 Inheritance diagram of gene input formatting classes

The input formatting of gene sequence in Hadoop is designed as follows:

- Define *GeneRecordFormat* class, which inherits from *FileInputFormat* class and override *createRecordReader*() and *isSplitable*() methods; Return *GeneLineRecordReader* object in *createRecordReader*() method which can receive a parameter of delimiter.
- Define *GeneRecordReader* class, which inherits from *RecordReader* class. Use *NextKeyValue*() method in the source code of *LineRecordReader* class to read key-value pairs in data files. Run method *nextKeyValue*() once to read a line of data files by invoking the *readLine*() method. Design a new *nextKeyValue*() method in *GeneRecordReader*, allowing the method to read multiple lines of data files every time it runs, such as *Algorithms* 1 and 2.
- Define *GeneLineReader* class, which implements *Closeable* interface. In the *LineReader* source code, there are many data line reading methods, namely, *readLine*(), *readCustomLine*(), *readDefaultLine*(), which empty the cache after reading one line of content, making it impossible to accumulate multiple lines of content. Thus, design a new reading method in the *GeneLineReader* class, which reads multiple lines without emptying the cache until completing the reading of multi-line content of a sequence. Use a flag variable, whose values are passed in by the input formatting algorithm in the *nextKeyValue*() method, to determine whether to empty the cache.
- Define *GeneSplitLineReader* class, which inherits from *GeneLineReader* class, and create constructor and *needAdditionalRecordAfterSplit*() method.
- Define *GeneCompressedSplitLineReader* and *GeneUncompressedSplitLineReader* classes, which inherit from *GeneSplitLineReader*, and use them for data compression.

The gene input formatting algorithm should be compatible with the running environment of the original program. By analyzing the source code, the author summarized the requiremetns on the data input algorithm in the new class: input formatting algorithms in *nextKeyValue*() Method of *GeneRecordReader* Class, invoke method *nextKeyValue*() once to read a gene record, clear the cache before reading a new gene record, keep the cache when reading the lines in a gene record, invoke method *nextKeyValue*()once and call *readLine*() many times, and keep the number of lines in a gene record constant or identifiable by features.

Considering these requirements, two input formatting algorithms were designed to fit in with more formats of gene sequence files: the input formatting algorithm based on line features (*Algorithm* 1) and the input formatting algorithm based on specified number of lines (*Algorithm* 2).

Specifically, *Algorithm* 1 was set up based on identifiable features of data. For example, in a FASTQ file, a gene record has a distinct identifiable feature. The first and end lines respectively starts with "@" and "!". The Boolean variable *flag* in the algorithm determines whether the cache is emptied (the cache is emptied when the variable is *true* and retained when the variable is *false*). The variable was initialized as *true* and set to *false* when the first line of a record was read, allowing the multi-line content of a record to accumulate into the *value* variable. To identify the features of new line, the *lastvalue* variable was defined to record the values before *value* accumulation. This variable can locate the new line of *value* and solve the problem of duplicate or empty lines. The multiple lines of a record were read by invoking *readLine*() method in a loop, and the start and end of the loop were controlled through line feature identification. Thus, *Algorithm* 1 can recognize the record clearly, even if the number of lines varies with records.

Algorithm 2 was designed based on neat record format. The specified number of lines was read in a loop, forming a record. The reading errors will occur if there are blank lines or inconsistent formats in the data file. The flag variable plays the same role as in *Algorithm* 1. The cache content is retained after reading the first line of the record.

Algorithm 1. Input formatting algorithm based on line features

```
flag = true  // Cache empty flag
lastvalue = ""  // Record cache before update
value = "" // Record cache
while(true){
  value += readLine(flag)  // While flag is true, clear the cache before reading
  if (value.length() > lastvalue.length()){
        Extracting features of new line
        Lastvalue = value; // update lastvalue
  }
  if (Satisfying the first line judgment condition)
        flag = false
  if (Satisfying the last line judgment condition)
        break
}
```

Algorithm 2. Input formatting algorithm based on specified number of lines

```
flag = true // Cache empty flag
value ="" // Record cache
//rowCount is count of lines per record
for (i = 1 to rowCount){
  value += readLine(flag)  // While flag is true, clear the cache before reading
  if (i == 1)  // Begin to retain the cache
    flag = false;
}
```

Obviously, the two algorithms were designed for data files with different features. Each of them has its own merits and defects. Based on the features of record lines, *Algorithm* 1 can effectively control blank lines and duplicate lines, but cannot read non-characteristic lines. Neither can it read records in different formats, before adjusting the condition of feature judgment. This algorithm is generally suitable for data files with different number of lines, if each line

contains identifiable features. With a constant number of lines, Algorithm 2 cannot solve the problem of empty and repeated lines. To fit different record formats, the number of loops must be adjusted. This algorithm applies to data files with neat format, i.e. the number of lines per record is fixed with no blank line.

MapReduce computing model

In this paper, the Hadoop computing model is designed based on the functions and programming rules of the platform. After input formatting, the task goes through the following five phases: Map, Data partitioning, Merging, Reduce and Alignment [12]. Below are the details on each phase.

In the Map phase, a tag is added to the pair-end sequence, indicating which end the sequence belongs to, but not added to the single-end sequence. After adding the end-indicator, the left- and right-ends of the pair-end sequence form a pair with the same key, laying the basis for subsequent identification. The processing in this phase is summarized as *Algorithm* 3.

Algorithm 3. Map algorithm

```
INPUT: (key, value)
OUTPUT: (key, value')
if (value stores a single-end sequence)
        value' = value
if (value stores a pair-end sequence){
        if (value comes from the left-end sequence file)
                value' = value + left-end tag
        if (value comes from the right-end sequence file)
                value' = value + right-end tag
}
Context.write(key, value')
```

In the phase of data partitioning, the partitioning algorithm embedded in Hadoop can distribute data according to the demand [8]. Note that the gene sequence alignment is under the following constraints: the number of left-end sequences equals that of right-end sequences in each partition; the sequences in a partition remains in the same order for sequence files; the left- and right-ends of a pair-end sequences must be allocated to the same partition. The processing in this phase is summarized as *Algorithm* 4.

Algorithm 4. Partitioning algorithm

```
INPUT: (key, value')
OUTPUT: partitionNum
fileSize = left-end seq file' size //the same as right-end seq file
partitionNum = 0
while (partitionNum < n){
        if (key < (partitionNum + 1)fileSize/n)
                return partitionNum
        partitionNum++
}
return n – 1
```

Firstly, the offset interval is computed by the size of sequence file and the number of data partitions. Then, the partition number is calculated with the key of the key-value pairs, which represents the position of a sequence in the sequence file. Meanwhile, the partition number is obtained in light of the offset interval of the key. In addition, the left- and right-ends of the pair-end sequence are assigned to the same partition, as they have the same key at the same position.

In the merging phase, the functions embedded in Hadoop are adopted to shuffle, sort and group the data in a partition according to the key. In this way, the sequences with the same key are merged into the same dataset. For a pair-end sequence, a key corresponds to two values, and the form of key-value pair is like (*key*, *value*' []). In each partition, all sequences are sorted to remain in the same order for the sequence file.

In the Reduce phase, the reducer function receives a data partition aggregated by key. If the input is a single-end sequence, all sequences in the partition are written to the local file. If the input is a pair-end sequence, the sequence is written to different local files according to the end-indicator. This process is summed up as *Algorithm* 5. At the end of this phase, a single-end sequence outputs a single local file, while a pair-end sequence outputs two local files.

Algorithm 5. Reduce algorithm

```
INPUT: (key, value'[])
OUTPUT: left_part_i, right_part_i OR part_i
//Output is local file, i represents node number
for ( value: value'[]){
        if (value has a left-end tag)
                MutiOutput.write("left", value)
        else if (value has a right-end tag)
                MutiOutput.write("right", value)
        else
                Context.write(value)
}
```

In the alignment phase, the sequence alignment task is initiated in the *cleanup*() function of the reducer and invoked by shell command, after the generation of the local data file(s). The alignment task is performed in parallel on each node in the cluster. The single- and pair-end alignment algorithms are launched to process the single and pair-end sequences, respectively. After the completion of the tasks on a node, the result files are stored in the HDFS. When the alignment tasks of all nodes are completed, the multiple result files in the HDFS are merged into one file.

Results and discussion

Experimental design

The following experiments were designed to verify the performance of our gene sequence input formatting method and MapReduce computing model. The gene data were extracted from the 1000 Genome Projects [15]. The 3.3G GRCh38.p12_genomic.fna was taken as the reference genome, while two datasets, ERR000589 and SRR062634, were selected as the short reads sequence. The specific information of the sequence is shown in Table 4.

Table 4. Short reads sequence datasets

Tag	Name	Number of reads	Read length, [bp]	Size, [GB]
D1	NA12750/ERR000589	1.2×10^7	51	5.2
D2	HG00096/SRR062634	6.7×10^6	200	3.5

As shown in Table 4, D1 is composed of pair-end sequences with a length of 51bp. It is suitable for BWA backtrack algorithm. D2 is composed of single-end sequences with a length of 200 bp. It is suitable for BWA MEM algorithm. The two datasets differ in size, sequence length, sequencing method and alignment algorithm, indicating that the datasets were selected appropriately.

The test cluster is a Hadoop cluster of one name node and eight data nodes. Each node is a VMware virtual machine with 8-core CPU, 8G memory and 1T hard disk. The Hadoop uses the version of 2.7.3. The operating system is Red Hat Enterprise Linux 6.5. During the experiments, the BWA mapping was performed with D1 and D2 as inputs. The same computing tasks were run on Hadoop clusters with 1, 2, 4, 6, and 8 work nodes, respectively. The time consumption, speedup and efficiency of each task were measured to evaluate the model performance.

Results analysis

All tasks were completed smoothly. The experimental results were the same as those of single-machine operation, but achieved at a much shorter time. This means the gene sequence input formatting method is valid and stable, and the computing model based on this method enjoys a good performance. Table 5 shows the time consumption, speedup and efficiency of our model on D1 and D2 at different number of work nodes. It can be seen that, despite the different amounts of data, the two tasks consumed a similar length of time on the same number of nodes. As shown in Figs. 3-5, the number of nodes is positively correlated with the time consumption and speedup of the two tasks and negatively with the efficiency of parallel computing. The experimental data show that the computing model greatly reduced the time consumption for datasets of different sizes and different alignment algorithms, and improved the speedup ratio and parallel computing efficiency.

Table 5. Performance comparison

Content	Dataset	Number of nodes				
		1	2	4	6	8
Time consumption, (min)	D1	258.8	131.1	67.8	45.3	38.5
	D2	249.6	127.1	72.8	49.5	36.7
Speedup	D1	1.0	2.0	3.8	5.7	6.7
	D2	1.0	2.0	3.4	5.0	6.8
Efficiency	D1	1.00	1.00	0.95	0.95	0.84
	D2	1.00	1.00	0.85	0.83	0.85

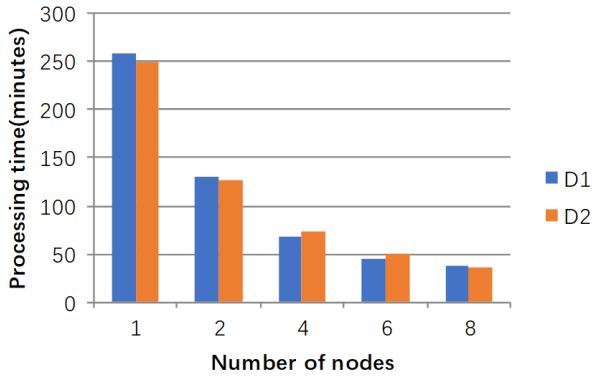

Fig. 3 Comparison of time consumption of D1 and D2

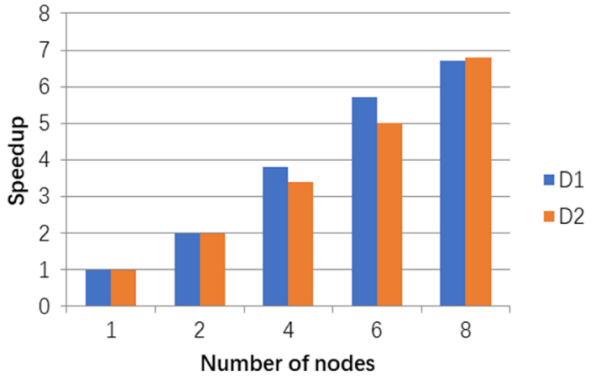

Fig. 4 Comparison of speedup of D1 and D2

The proposed computing model was further contrasted against several excellent parallel computing methods, which have been proved as capable of improving the performance of gene sequence alignment. All the experiments were performed using the same dataset on Hadoop clusters with different configurations. The grouping experiments were performed at 1, 2, 4, 6 and 8 nodes. Since these algorithms use different computing environments, the time consumption was not compared in the same dataset. In terms of the speedup ratio of parallel computing (Fig. 6), the proposed computing model had certain advantages over the contrastive algorithms. The results show that gene sequence input formatting method both saves the time of data preprocessing and reduces the burden of MapReduce computing model, improving the efficiency of parallel computing.

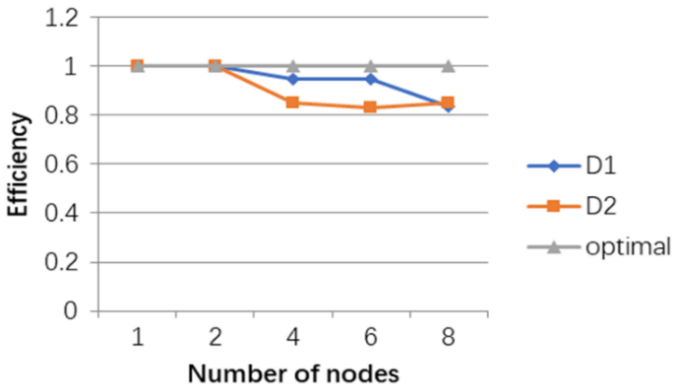

Fig. 5 Comparison of computing efficiency of D1, D2 and optimal

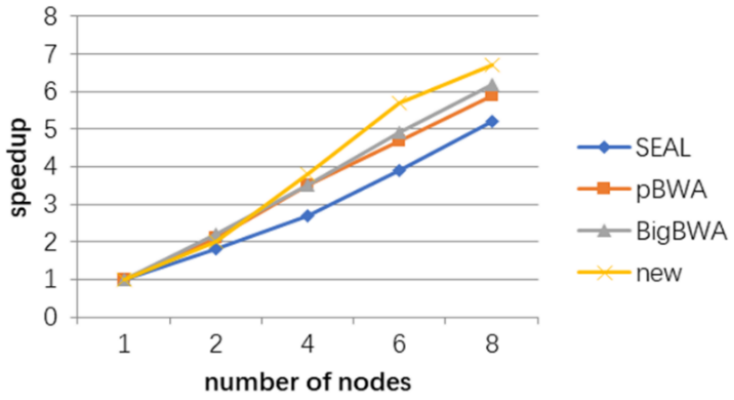

Fig. 6 Speedup comparison of different parallel methods

Conclusions

The proposed method of gene sequence input formatting can read many kinds of gene sequence files effectively on Hadoop, and transmit them to the computing model to complete the calculation. This method eliminates the need to preprocess data before submitting tasks, ensures the integrity and unity of Hadoop applications, and saves the time of interactive processing. It is safe to say that our method has some reference value for the record file reading in other fields.

Our MapReduce computing model, which is based on the above method, was proved experimentally to realize distributed parallel computing of gene sequence alignment. This model makes gene sequence alignment scalable in distributed cluster, and significantly reduces the time consumption of the same task from the level of single computer. In addition, the model has a better speedup ratio of parallel computing than other gene sequence parallel computing approaches.

For similar tasks, the parallel computing plan can be designed according to our integrated solution: the input formatting classes should be designed according to Fig. 2, the input algorithm should be selected from *Algorithms* 1 and 2 for input formatting; the MapReduce computing model should be designed with reference to *Algorithms* 3, 4 and 5.

Acknowledgements

This work was supported by National Natural Science Foundation of China project 61462070.

References

1. Abuín J. M., J. C. Pichel, T. F. Pena, J. Amiqo (2015). BigBWA: Approaching the Burrows-Wheeler Aligner to Big Data Technologies, Bioinformatics, 31(24), 4003-4005.
2. Almeida J. S., A. Grüneberg, W. Maass, S. Vinga (2012). Fractal MapReduce Decomposition of Sequence Alignment, Algorithms for Molecular Biology, 7(1), 1-12.
3. Cock P. J., C. J. Fields, N. Goto, M. L. Heuer, P. M. Rice (2009). The Sanger FASTQ File Format for Sequences with Quality Scores and the Solexa/Illumina FASTQ Variants, Nucleic Acids Research, 38(6), 1767-1771.
4. Dean J., S. Ghemawat (2008). MapReduce: Simplified Data Processing on Large Clusters, Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, 51(1), 107-113.
5. Decap D., J. Reumers, C. Herzeel, P. Costanza, J. Fostier (2015). Halvade: Scalable Sequence Analysis with MapReduce, Bioinformatics, 31(15), 2482-2488.

6. Gattiker A., F. H. Gebara, H. P. Hofstee, J. D. hayes, A. hylick (2013). Big Data text-oriented Benchmark Creation for Hadoop, IBM Journal of Research & Development, 57(3/4), 1-6.
7. Ghoneimy S., S. A. El-Seoud (2016). A MapReduce Framework for DNA Sequencing Data Processing, International Journal of Recent Contributions from Engineering, Science & IT, 4(4), 11-20.
8. Gufler B., N. Augsten, A. Reiser, A. Kemper (2012). The Partition Cost Model for Load Balancing in MapReduce, Cloud Computing and Services Science, 371-387.
9. Apache Hadoop Software Library, http://hadoop.apache.org/ (Last access 20 June 2018).
10. Li H. (2009). The Sequence Alignment / Map (SAM) Format, Bioinformatics, 25(1 Pt 2), 1653-1654.
11. Metzker M. L. (2010). Sequencing Technologies – the Next Generation, Nature Reviews Genetics, 11(1), 31-46.
12. Pandey R. V., S. Christian (2013). DistMap: A Toolkit for Distributed Short Read Mapping on a Hadoop Cluster, PLOS ONE, 8(8), e72614.
13. Pireddu L., S. Leo, G. Zanetti (2011). SEAL: A Distributed Short Read Mapping and Duplicate Removal Tool, Bioinformatics, 27(15), 2159-2160.
14. Schatz M. C. (2009). CloudBurst: Highly Sensitive Read Mapping with MapReduce, Bioinformatics, 25(11), 1363-1369.
15. Watson J. D. (1990). The Human Genome Project: Past, Present and Future, Science, 248(4951), 44-49.

Xiaolong Feng, Ph.D. Student
E-mail: fengxl@imau.edu.cn

Xiaolong Feng, Ph.D. candidate. His main research directions are big data processing and bioinformatics computing.

Prof. Jing Gao, Ph.D.
E-mail: gaojing@imau.edu.cn

Jing Gao, Ph.D., Professor, mainly engaged in cloud computing, big data and agricultural informatization research.