Spark-based Parallelization of Basic Local Alignment Search Tool

Hui Wang[1,2], Leixiao Li[1,2*], Chengdong Zhou[1,2], Hao Lin[1,2], Dan Deng[1,2]

[1]*College of Data Science and Application*
Inner Mongolia University of Technology
Hohhot 010080, China
E-mails: 122700185@qq.com, llxhappy@126.com, 1452082002@qq.com, suzukaze_aoba@foxmail.com, dengdan2020@163.com

[2]*Inner Mongolia Autonomous Region Engineering & Technology Research Center of Big Data Based Software Service*
Inner Mongolia University of Technology
Hohhot 010080, China

[*]*Corresponding author*

Received: September 03, 2019 **Accepted: February 21, 2020**

Published: March 31, 2020

Abstract: *Sequence alignment is a key link of bioinformatics analysis. The basic local alignment search tool (BLAST) is a popular sequence alignment algorithm with high accuracy. However, the BLAST is inefficient in comparing and analyzing a massive amount of gene sequencing data. To solve the problem, this paper designs a distributed parallel BLAST method called SparkBLAST, based on the big data technique Spark. Under the in-memory computing framework Spark, SparkBLAST identifies the task of sequence alignment, divides the sequence dataset, and compares the sequence data. The Apache Hadoop YARN was adopted to task scheduling and resource allocation. Finally, the SparkBLAST was compared with standalone BLAST through experiments. The results show that SparkBLAST realized the speedup ratio of 3.95, without sacrificing the accuracy. In other words, SparkBLAST greatly outshines the standalone BLAST in calculation efficiency. The research findings provide bioinformatics researchers a highly efficient tool for sequence alignment.*

Keywords: *Sequence alignment, Basic local alignment search tool, Spark, Parallelization, Speedup.*

Introduction

In the field of bioinformatics, the gene sequencing data are growing at an exponential rate, thanks to the next-generation sequencing technology. As a result, sequence alignment has become the key link of sequence analysis. One of the most popular algorithms for sequence alignment is the basic local alignment search tool (BLAST) [1]. However, the BLAST is inefficient in comparing and analyzing a massive amount of gene sequencing data. The low efficiency cannot satisfy the surging demand for the analysis of genetic data, slowing down the progress in bioinformatics.

The BLAST algorithm has been repeatedly improved from the perspective graphics processing unit (GPU). In 2011, Vouzis and Sahinidis [20] modified the BLAST algorithm based on GPU technology, and created the GPU-BLAST algorithm, which is 3 times more efficient than the BLAST algorithm. In 2017, Ye et al. [22] developed the heterogeneous BLAST (H-BLAST) algorithm with 2 GPU threads, and verified that the H-BLAST is faster than the 16-thread BLASTX, which is developed by the National Center for Biotechnology Information (NCBI). In addition, the H-BLAST was found to be 1.5 to 4 times faster than

GPU BLAST. In 2011, Liu et al. [13] proposed the protein-protein BLAST (BLASTP) under compute unified device architecture (CUDA) based on GPU technology, and proved that the CUDA-BLASTP achieved a speedup of 2-3 times. In 2017, Khare et al. [9] created a new pattern search for DNA sequences called HCUDABLAST, which combines the multi-core parallelism of the GPU with the scalability of the Hadoop framework. In 2017, Rani and Gupta [19] put forward the CLUS-GPU-BLASTP algorithm. On a single computing node, the CLUS-GPU-BLASTP is 2.1 times faster than the GPU-BLAST; on a cluster of 12 computing nodes, the CLUS-GPU-BLASTP is 13.2 times faster than the latter. Moreover, the CLUS-GPU-BLASTP is 7.4-8.2 times faster than the standard single-thread NCBI-BLAST.

In recent years, many have attempted to improve the BLAST algorithm based on distributed design, big data technology and cloud computing. In 2002, Bjornson et al. [3] proposed the TurboBLAST algorithm with a distributed design, which relies on distributed computing to realize efficient calculation. In 2009, Matsunaga et al. [14] developed the CloudBLAST method on a cloud computing platform. In the CloudBLAST, virtual machines (VMs) and virtual network cloud platforms are operated under the MapReduce computing framework, while subtask division is conducted to realize the parallel computing of the BNC2 gene provided by the NCBI. In 2011, Kent [8] invented the BLAST-like in the spirt of distributed computing: (1) The large task file and index database file of the large dataset are divided into blocks of the same size; (2) The query sequence is constructed into a word list; (3) The word list is scanned and compared in details. In 2012, Yang et al. [21] implemented the MapReduce-BLAST [10] algorithm based on Hadoop platform, and realized an efficient and scalable sequence alignment platform. In 2013, Meng et al. [15] created an efficient and reliable parallel BLAST algorithm under the Hadoop-based MapReduce computing framework.

To sum up, there are two ways to improve the BLAST algorithm: GPU-based improvement and big data-based parallelization. The GPU-based improvement greatly enhances the computing efficiency. However, few institutes can afford to buy the costly GPUs required for the improvement. The big data-based parallelization is easy and cheap to implement, because the distributed design has a relaxed requirement on hardware. But the parallelization mostly takes place under the MapReduce computing framework. This disk-based framework has input/output (IO) limitations, and faces a low computing efficiency, if the disks are read and written frequently.

To overcome the defects of the above methods, this paper proposes a distributed parallel BLAST algorithm called SparkBLAST, based on Spark [2, 13]. Under the in-memory computing framework Spark, SparkBLAST identifies the task of sequence alignment, divides the sequence dataset, and compares the sequence data. The Apache Spark YARN [17] was adopted to task scheduling and resource allocation. Finally, the SparkBLAST was compared with standalone BLAST through experiments. The results show that the proposed algorithm can improve the computing efficiency, without sacrificing the accuracy [6]. The research results greatly promote the development of bioinformatics.

Spark-based parallelization of BLAST

Spark is an open-source distributed computing framework developed at the University of California, Berkeley's AMP Lab. There are multiple advantages of this famous framework. First, Spark enables in-memory data sharing across Directed Acyclic Graph (DAG). Second, Spark adopts a memory cache mechanism, which reduces frequent IO reads and writes and improves computing efficiency. Third, Spark supports many operating modes, and its

resources can be managed by YARN and Apache Mesos [16]. Fourth, Spark supports various distributed file systems, e.g. Hadoop distributed file system (HDFS) and the Cassandra file system (CFS) [12]. Fifth, Spark is compatible with multiple languages, namely, Scala, Python, and Java.

Official tests show that Spark has a much better computing framework than MapReduce: Spark in-memory computing is more than 100 times faster than MapReduce; even if disk-based computing is adopted, Spark remains 10 times more efficient than MapReduce. As a result, Spark-based parallel computing could achieve a high efficiency.

Based on NCBI's BLAST+ [4, 18], this paper develops a highly accurate distributed parallel BLAST algorithm, using the machine learning pipeline of Spark, and the Spark cluster. The proposed algorithm, denoted as SparkBLAST mainly consists of two parts: data preparation and parallel computing. Before data processing, the data format was unified as FASTA.

Data preparation

Two files should be inputted for comparison in BLAST: the target database file and a sequence alignment file. The former helps to set up the index database, while the latter contains multiple data on nucleic acid or protein sequence in FASTA format. In this research, the Spark cluster is composed of one master node (name node) and four slave nodes (data nodes). The name node manages the data of the cluster, while data nodes are responsible for data storage.

Before parallel computing, an index database of the reference genome was established on the local file system of the name node, and then distributed to the same directory of all data nodes in the cluster through remote secure copy (scp-r). Next, the sequence alignment file was uploaded to the HDFS as the input of parallel computing. As shown in Fig. 1, a data source is provided for parallel computing.

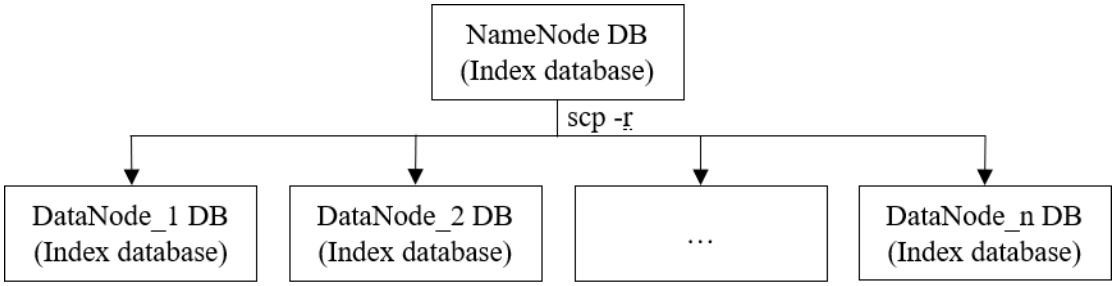

Fig. 1 Distribution of target database files

Distributed parallel computing of SparkBLAST

As shown in Fig. 2, the distributed parallel computing of SparkBLAST stores the sequencing data in HDFS in a scalable and distributed manner. The YARN serves as the cluster resource manager for task scheduling and resource allocation. The in-memory computing framework Spark is taken as the data processing framework to split the input sequence file into fragments. Meanwhile, the machine learning pipeline of Spark is adopted to call the external application BLAST+ to perform distributed parallel computing on the fragmented data. The computed results were combined and written to the local disk. Overall, the distributed parallel computing of SparkBLAST is implemented in four steps: task division, task allocation, parallel computing, and file merging.

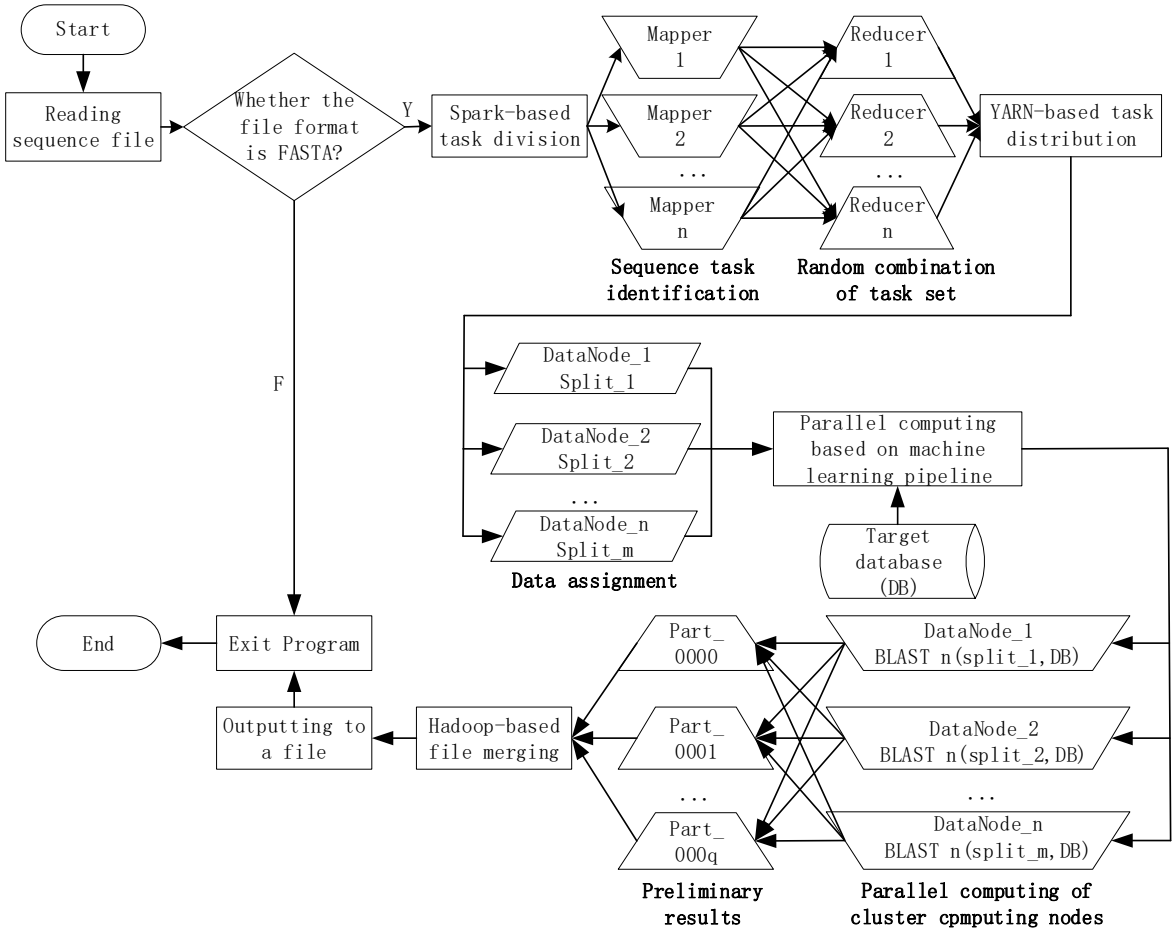

Fig. 2 Distributed parallel computing of SparkBLAST

Task division

First, the sequence file for comparison is uploaded to the HDFS via the user terminal by file transfer tools. Then, the sequence file serves as a shared data source for all computing nodes. Scala is chosen as the development language, for its high operating efficiency and performance on Spark. Before BLAST parallel computing, the Spark application driver is started programmatically. Then, the sequence file from HDFS is read, and the task of the sequence file is divided by the regular delimiter between sequences. The resulting subtasks are grouped into different task sets, according to the number of computing nodes in the cluster. The task sets are of the same size, because Spark adopts even partitioning by default. The main objective of this step is to split large tasks into fine-grained subtasks.

Task allocation

The YARN resource manager is adopted to allocate resources and distributes tasks. Before running Spark cluster, the operating mode of YARN is specified to set up a resource manager for cluster computing. Then, YARN will perform resource allocation and task distribution, according to the user-defined cluster parameters or the preset parameters of computing resources. In this step, the task sets obtained in the first step are distributed to each computing node in the cluster, preparing for parallel computing.

Parallel computing

The machine learning pipeline of Spark is started to pass the execution commands to the all computing nodes in Spark cluster, triggering the calculation script of the BLAST. Then, all computing tasks are initiated by calling the local application BLAST+ on each node.

Each task is executed in multiple processes, i.e. the set of subtasks is processed in parallel by the corresponding node. Finally, the result files are stored in the specified HDFS file directory.

File merging

The getmerge command of Hadoop is called to merge the result files of each computing node. The merged files are stored in the local file system of the master node, which are easy to view and download.

Advantages

This parallel computing method, denoted as SparkBLAST, enjoys the following advantages:

1) The parallel computing of the BLAST is realized without changing the NCBI's BLAST+ software application.
2) The HDFS of Hadoop cluster was taken to store the files, creating a highly reliable and scalable storage system for sequencing data.
3) The parallel computing of the BLAST was implemented under the in-memory computing framework Spark, providing a convenient and efficient tool for bioinformatics comparisons.
4) The analyst only needs to upload the comparison/query file through the user terminal. The parallel computing method is simple and highly automatic.
5) The flexible and versatile design supports all BLAST algorithms in BLAST+ software, namely, BLASTn, BLASTP [7] and BLASTX [5].

Comparative analysis

To verify its performance, the SparkBLAST with a four-node cluster was compared with the standalone BLAST through 10 experiments. The results of the two algorithms are contrasted in Fig. 3, where the speed is the mean value of the ten experiments. As shown in Fig. 3, when the data files were small, SparkBLAST was slightly less efficient than the standalone BLAST, for the Spark cluster spends time in network communication between data nodes. With the growing size of data files, however, SparkBLAST became increasingly superior to the standalone algorithm. For example, when the data files were about 1 GB in size, the computing efficiency of SparkBLAST was 3.95 times that of the standalone BLAST, close to the theoretical value of 4. The results show that the Spark cluster-based parallelization can greatly enhance the computing efficiency of the BLAST.

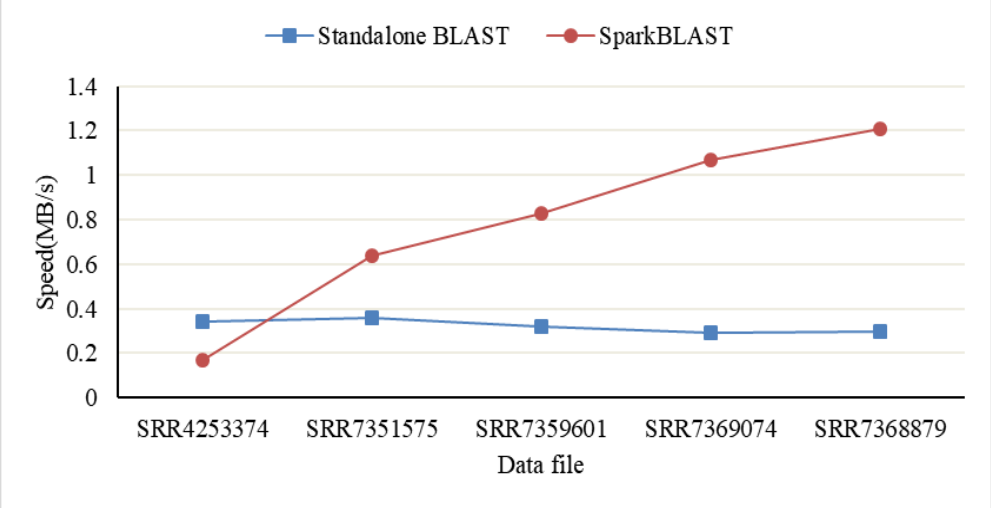

Fig. 3 Speed comparison between standalone BLAST and SparkBLAST

Sequence alignment experiments

Experimental data

For BLAST alignment experiments, the human gene sequencing data published by the NCBI were selected as the sequence file, and the human reference genome data (GCF_000001405.37_GRCh38.p11_genomic.fna, data size 3.2 GB) were taken as the target database file. The selected sequence files are of different sizes: 6.1 MB, 68 MB, 158.7 MB, 503.5 MB, and 1024 MB. For comparability, each sequence was controlled at 200 bp in length. For the 9.4 MB file, no data of the length of 200 bp was found in humans. Thus, the 300 bp was set as the length of that sequence. The details of each sequence file are given in Table 1 below.

Table 1. Details on sequence files

Name	Number of sequences, (strips)	Sequence length, (bp)	File size, (MB)
SRR4253374	32635	300	9.4
SRR7351575	341512	200	68.6
SRR7359601	793898	200	158.7
SRR7369074	2560486	200	503.5
SRR7368879	5102574	200	1024

Experimental environment

Our experiments were conducted on the VMware [10], a virtualized big data platform in the lab of Big Data Processing Center in Inner Mongolia Agricultural University. A total of five virtual machines (VMs) were deployed on the platform, acting as the computing nodes. The VMs operate in the 64bit Centos 6.5. The application software includes: NCBI's BLAST 2.6.0+, Hadoop 2.5.2, Spark 1.2.0, JDK 1.7 and Scala 2.10.6.

As such, the configurations of the VMs (computing nodes) in the Spark cluster is as follows:

Name node – Node 01, Node 02, Node 03, Node 04, Node 05;
Operating system – Centos 6.5; Memory – 8 GB; CPU – 4;
BLAST – BLAST 2.6.0+; Hadoop – Hadoop-2.5.2;
Spark – Spark-1.2.0; JDK – JDK-7u25.

Several experiments were conducted on the 5-node Spark cluster, and another was performed on the standalone BLAST.

Design of standalone experiment

For comparison, a VM with the same configuration as the five VMs was created, and deployed on the virtualized big data platform, creating the environment for the standalone BLAST experiment. Then, an operating system was installed to meet the requirements of the Spark cluster. Then, the BLAST+ 2.6.6, which is the same as that for the Spark cluster, was downloaded from the NCBI official website, and installed onto cluster nodes. Next, the authors downloaded high-throughput sequencing files and human reference genome files, which are consistent with those for the Spark cluster, from the NCBI official website. On this basis, a local target index database of the reference genome files was set up. The type of the database is denoted as "nucl", for the nucleic acid sequences were adopted for comparison. The command to build the index database is as follows:

```
#makeBLASTdb -in DB.fa -dbtype nucl -parse_seqids -out human
```

Furthermore, sequence files of different sizes were downloaded for the standalone BLAST experiment. The default parameters were used in the comparison command:

#BLASTn -db DB -outfmt 7 -num_threads 8 -query query.fasta -out output

The results of the standalone BLAST experiment are given in Table 2. The relationship between data file size and operating time is described in Fig. 4.

Table 2. Results of standalone experiment

Name	SRR4253374	SRR7351575	SRR7359601	SRR7369074	SRR7368879
File size (MB)	9.4	68.6	158.7	503.5	1024
Time (s)	28	193	495	1726	3344

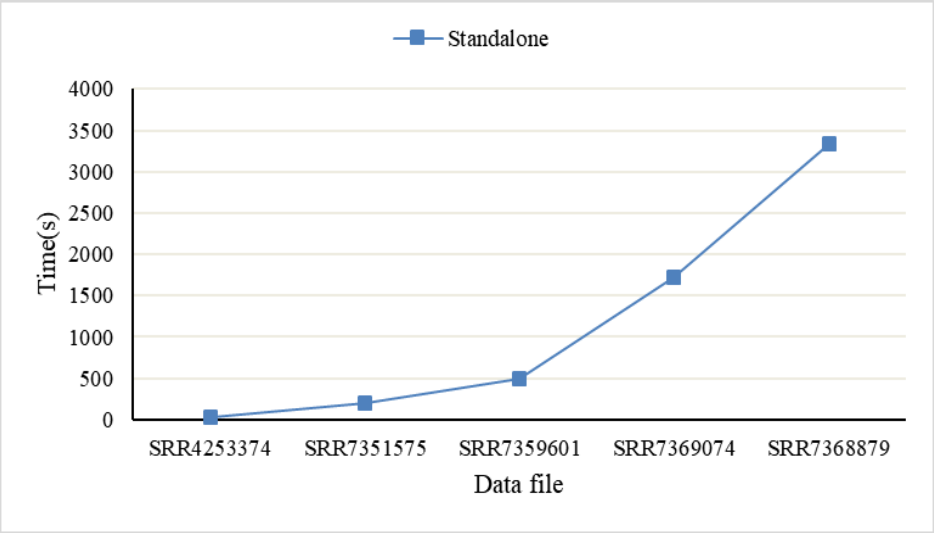

Fig. 4 The file size-time relationship of standalone experiment

Design of cluster experiments

First, the Hadoop and Spark cluster were started, and gene sequence alignment files of different sizes were uploaded to the HDFS. Then, an index database of human reference genome data files was created, and the files in the index database were distributed to the same local directory of each computing node for sequence alignment. After that, the comparison task was submitted to the Spark cluster via the user terminal for parallel comparison of BLAST [11]. The submission commands are as follows:

#Spark-submit --master yarn-client --executor-memory $executor_memory --driver-memory $driver_memory --num-executors $num_executors --executor-cores $executor_cores --driver-cores $driver_cores --class SparkBLAST /SparkBLAST.jar $splits_num "BLASTn -db /target_DB -outfmt 7 -num_threads 8 query_file out_file; hdfs dfs -getmerge out_file local_file.

The results of the SparkBLAST experiments are given in Table 3. The relationship between data file size and operating time is described in Fig. 5.

Table 3. Results of cluster experiments

Name	SRR4253374	SRR7351575	SRR7359601	SRR7369074	SRR7368879
File size, (MB)	9.4	68.6	158.7	503.5	1024
Time, (s)	55	108	190	471	846

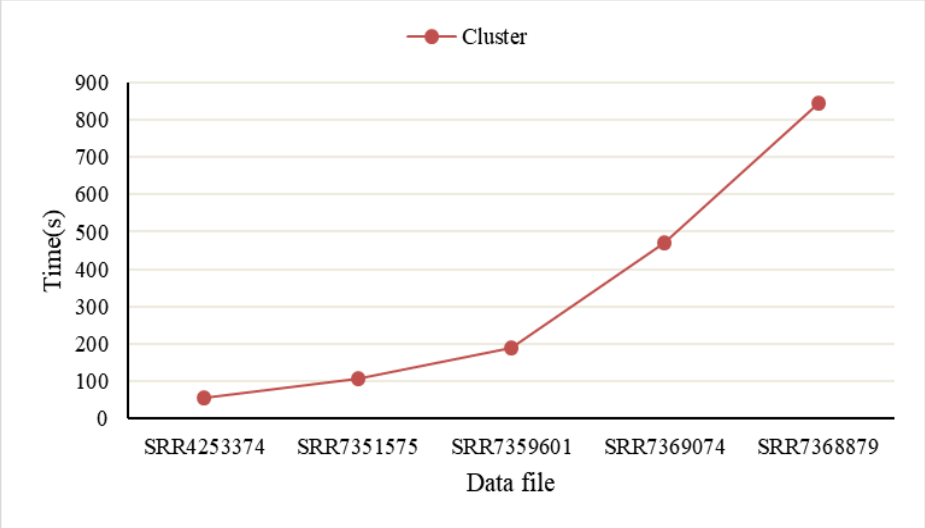

Fig. 5 The file size-time relationship of cluster experiments

Results comparison

Both standalone experiment and each cluster experiment were repeated ten times. Table 4 compares the results of the standalone experiment and those of cluster experiments. The results are the mean values of the ten repeated experiments.

Table 4. Comparison of results between standalone experiment and cluster experiments

Name	SRR4253374	SRR7351575	SRR7359601	SRR7369074	SRR7368879
File size, (MB)	9.4	68.6	158.7	503.5	1024
Time of standalone, (s)	28	193	495	1726	3344
Time of cluster, (s)	55	108	190	471	846
Speed of standalone, (MB/s)	0.34	0.36	0.32	0.29	0.30
Speed of cluster, (MB/s)	0.17	0.64	0.83	1.07	1.21

Fig. 6 compares the file size-time relationships between standalone and cluster experiments. With the growing size of sequence file, the Spark cluster consumed a much shorter time to process the same dataset than the standalone BLAST. At the beginning, the standalone BLAST and SparkBLAST consumed basically the same amount of time. With the increase in file size, the time consumed by standalone BLAST was 3.95 times that of SparkBLAST, close to the theoretical value of 4. Hence, SparkBLAST is significantly more efficient than the standalone BLAST, when the dataset is large.

Fig. 7 presents the speedup ratio of each data file. It can be seen that the SparkBLAST is much more efficient than the standalone BLAST. With the growth in file size, the speedup ratio gradually reached 3.95. Thus, our parallel design provides an efficient and fast BLAST application for sequence comparison.

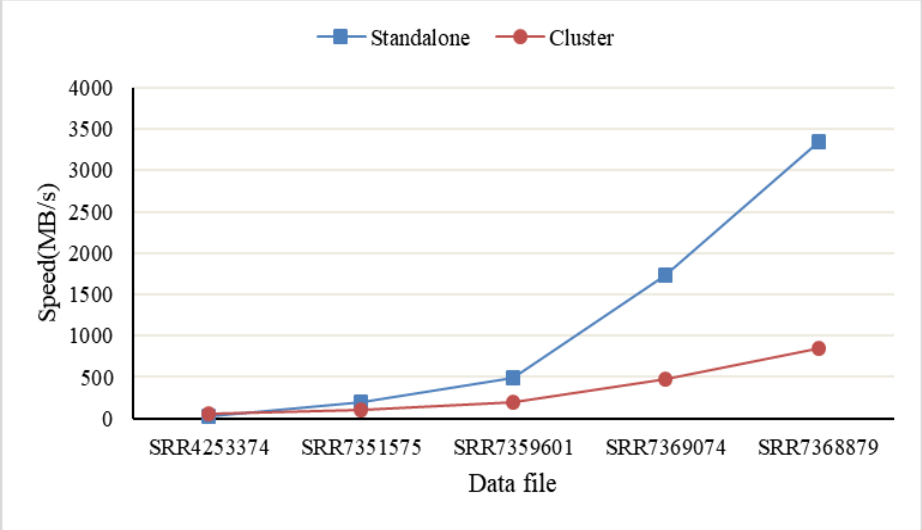

Fig. 6 Comparison between the file size-time relationships of standalone experiment and cluster experiments

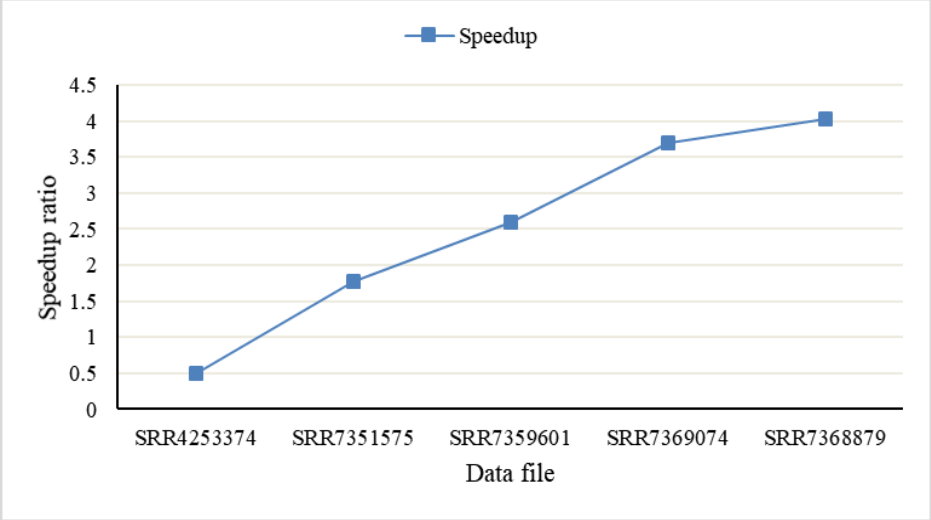

Fig. 7 The relationship between data size and speedup ratio

Next, the amounts of data processed by the standalone BLAST and SparkBLAST per unit time were computed based on the experimental results, as a sign of the speed of data processing. Fig. 8 compares the processing speeds of the standalone and cluster experiments. The comparison clearly shows that the speed of the standalone BLAST remained basically the same, while that of SparkBLAST increased significantly with the growing data size. Overall, the SparkBLAST greatly outperformed the standalone BLAST in speed when processing largescale data, but had no significant advantage when processing small scale data.

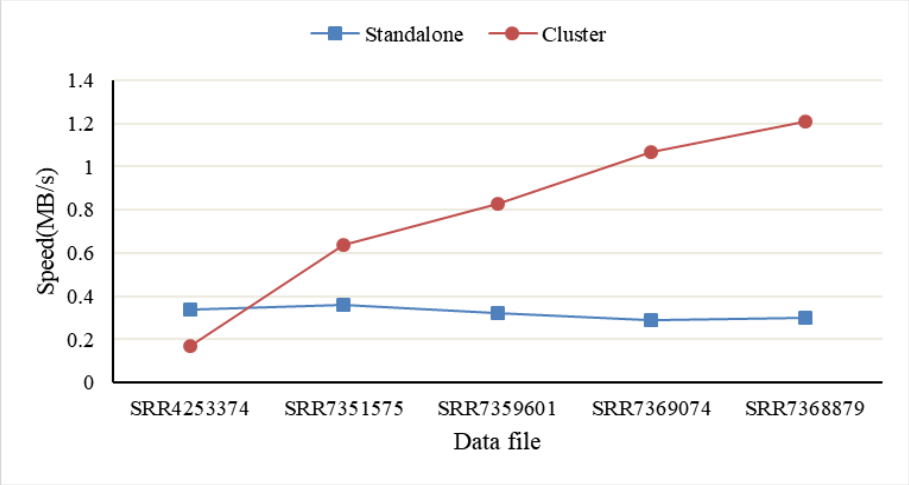

Fig. 8 Comparison between the speed-data size relationships of standalone experiment and cluster experiments

Conclusions

Based on Spark cluster, this paper designs a parallel BLAST algorithm for sequence alignment. The sequence data were stored in the HDFS, a distributed storage system with high scalability, reliability and availability. The proposed SparkBLAST algorithm was compared with the standalone BLAST through experiments. The comparison shows that the SparkBLAST had obvious advantages in processing largescale data, but the advantages were not obvious when processing small scale data. This is because SparkBLAST spends lots of time in communication between computing nodes in the cluster, when it computes small data files. Our algorithm provides important new insights into parallel and scalable analysis of genetic data.

Acknowledgements

The work is funded in part by the Inner Mongolia Natural Science Foundation Project under Grant No. 2019MS06027, Inner Mongolia Key Technology Research Plan Project (Toward Big Data Storage and Mining Platform for Intelligent Transportation), Major Special Project of Science and Technology in Inner Mongolia Autonomous Region (Development and Application of Private Cloud Operating System Based on OpenStack).

References

1. Altschul S. F., W. Gish, W. Miller, E. W. Myers, D. J. Lipman (1990). Basic Local Alignment Search Tool, Journal of Molecular Biology, 215(3), 403-410.
2. Awan A. J., M. Brorsson, V. Vlassov, E. Ayguade (2016). Architectural Impact on Performance of In-memory Data Analytics: Apache Spark Case Study, arXiv Preprint arXiv:1604.08484.
3. Bjornson R. D., A. H. Sherman, S. B. Weston, N. Willard, J. Wing (2002). TurboBLAST®: A Parallel Implementation of BLAST built on the TurboHub, Proceedings of International Parallel and Distributed Processing Symposium, 1-8.
4. BLAST+, https://blast.ncbi.nlm.nih.gov/Blast.cgi, (Last access March 15, 2020).
5. Huson D. H., C. Xie (2014). A Poor Man's BLASTX – High-throughput Metagenomic Protein Database Search Using Pauda, Bioinformatics, 30(1), 38-39.
6. Islam N. S., M. Wasi-ur-Rahman, X. Lu, D. Shankar, D. K. Panda (2015). Performance Characterization and Acceleration of In-memory File Systems for Hadoop and Spark Applications on HPC Clusters, Proc. of the 2015 IEEE Int Conf on Big Data, 243-252.

7. Jacob A., J. Lancaster, J. Buhler, B. Harris, R. D. Chamberlain (2008). Mercury BLASTP: Accelerating Protein Sequence Alignment, ACM Transactions on Reconfigurable Technology & Systems, 1(2), 9-16.
8. Kent W. J. (2002). BLAT – The BLAST-like Alignment Tool, Genome Research, 12(4), 656-664.
9. Khare N., A. Khare, F. Khan (2017). HCudaBLAST: An Implementation of BLAST on Hadoop and Cuda, Journal of Big Data, 4(1), 41.
10. Kuminsky K. (2015). VMware vCenter Cookbook, Packt Publishing Ltd.
11. Ladunga I. (2017). Finding Similar Nucleotide Sequences Using Network BLAST Searches, Current Protocols in Bioinformatics, 3.3.1-3.3.26.
12. Lakshman A., P. Malik (2010). Cassandra: A Decentralized Structured Storage System, ACM SIGOPS Operating Systems Review, 44(2), 35-40.
13. Liu W., B. Schmidt, W. Muller-Wittig (2011). CUDA-BLASTP: Accelerating BLASTP on CUDA-enabled Graphics Hardware, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 8(6), 1678-1684.
14. Matsunaga A., M. Tsugawa, J. Fortes (2008). Cloudblast: Combining MapReduce and Virtualization on Distributed Resources for Bioinformatics Applications, Proceedings of the 2008 IEEE Fourth International Conference on eScience, 222-229.
15. Meng M., J. Gao, J. J. Chen (2013). Blast-parallel: The Parallelizing Implementation of Sequence Alignment Algorithms Based on Hadoop Platform, Proceedings of the 2013 6th International Conference on Biomedical Engineering and Informatics, 465-470.
16. Meyerson J. (2016). Ben Hindman on Apache Mesos, IEEE Software, 33(1), 117-120.
17. Murthy A. C., V. K. Vavilapalli, D. Eadline (2014). Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2, Pearson Education.
18. NCBI, https://www.ncbi.nlm.nih.gov , (Last access March 15, 2020).
19. Rani S., O. P. Gupta (2017). CLUS_GPU-BLASTP: Accelerated Protein Sequence Alignment Using GPU-enabled Cluster, J Supercomput, 73(10), 4580-4595.
20. Vouzis P. D., N. V. Sahinidis (2011). GPU-BLAST: Using Graphics Processors to Accelerate Protein Sequence Alignment, Bioinformatics, 27(2), 182-188.
21. Yang X. L., Y. L. Liu, C. F. Yuan, Y. H. Huang (2011). Parallelization of BLAST with MapReduce for Long Sequence Alignment, Proceedings of Fourth International Symposium on Parallel Architectures, Algorithms and Programming, 241-246.
22. Ye W., Y. Chen, Y. Zhang, Y. Xu (2017). H-BLAST: A Fast Protein Sequence Alignment Toolkit on Heterogeneous Computers with GPUs, Bioinformatics, 33(8), 1130-1138.

Hui Wang, M.Sc.
E-mail: 122700185@qq.com

Hui Wang is a Lecturer at the College of Data Science and Application, Inner Mongolia University of Technology, China. She has a Master's Degree in Computer Application Technology from Harbin Engineering University in China. Her research interests include data mining, cloud computing, and big data processing.

Prof. Leixiao Li
E-mail: llxhappy@126.com

Leixiao Li is a Professor at the College of Data Science and Application, Inner Mongolia University of Technology, China. Prof. Li has a Master's Degree in Computer Application Technology from Inner Mongolia University of Technology, China. His research interests include data mining, cloud computing and big data processing, etc.

Chengdong Zhou, M. Sc. Student
E-mail: 1452082002@qq.com

Chengdong Zhou is a graduate student at the College of Data Science and Application, Inner Mongolia University of Technology, China. His research interests include data mining, cloud computing and big data processing, etc.

Hao Lin, M. Sc. Student
E-mail: suzukaze_aoba@foxmail.com

Hao Lin is a graduate student at the College of Data Science and Application, Inner Mongolia University of Technology, China. His research interests include data mining, cloud computing and big data processing, etc.

Dan Deng, M. Sc. Student
E-mail: dengdan2020@163.com

Dan Deng is a graduate student at the College of Data Science and Application, Inner Mongolia University of Technology, China. His research interests include data mining, cloud computing and big data processing, etc.