DGCF: A Distributed Greedy Clustering Framework for Large-scale Genomic Sequences

Zekun Yin
School of Software
Shandong University, Jinan, China
Shenzhen Research Institute of Shandong University
zekun.yin@mail.sdu.edu.cn

Xiaoming Xu
School of software
Shandong University
Jinan, China
xiaoming.xu@mail.sdu.edu.cn

Kaichao Fan
School of Software
Shandong University
Jinan, China
kaichao.fan@mail.sdu.edu.cn

Ruilin Li
Computer Network Information Center
Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
lirl@sccas.cn

Weizhong Li
J. Craig Venter Institute
La Jolla, CA 92037, USA
wli@jcvi.org

Weiguo Liu*
School of software
Shandong University, Jinan, China
National Supercomputing Center in Wuxi, China
weiguo.liu@sdu.edu.cn

Beifang Niu*
Computer Network Information Center
Chinese Academy of Sciences
University of Chinese Academy of Sciences
Beijing, China
niubf@cnic.cn

***Abstract*—Clustering is a very fundamental while time-consuming compute operation in biological sequence analysis. New sequencing technologies such as NGS and 3GS have dramatically increased both the dataset size and the length of a single read sequence. However, existing tools lack scalability for handling large-scale datasets as well as long sequences. A feasible solution to this problem is to use parallel and distributed systems. The efficient deployment of such systems, however, requires high parallelism in both software implementations as well as algorithmic optimizations. In this paper, we propose DGCF, a Distributed Greedy Clustering Framework which is capable to handle large-scale datasets and long sequences. Our framework adopts a greedy clustering strategy which overlaps communication with computation among many distributed computing nodes. We also design and implement a sparse suffix array (SSA)-based alignment algorithm that can support long sequences. Experiments show that our framework achieves near-linear speedups on a distributed memory cluster.**

***Index Terms*—greedy clustering, sparse suffix array, sequence analysis, parallel computing**

I. Introduction

Clustering analysis plays an important role in many biological analyses. Also, it is essential to reveal underlying natural data groups such as estimating species richness [1]. In the meanwhile, clustering can be used to reduce the database size by removing redundant sequences [2].

In the last decade, many clustering algorithms have been proposed to process biological sequences. There are two kinds of algorithms that have been widely used in this field. One is based on the greedy method. Popular tools such as CD-HIT [3], UCLUST [4], DNACLUST [5] and MMSEQS2 [6] adopted this method. The other one is based on hierarchical clustering which requires a similarity matrix constructed from the all-to-all pairwise alignment between sequences such as ESPRIT [1].

Hierarchical clustering starts by considering every sequence separately and merging the two closest ones in the similarity matrix into a cluster. Then the closest clusters or sequences join together iterative under a certain similarity threshold to construct larger clusters. This process is quite similar to the construction of a minimum spanning tree. And there are three different linkage criteria to measure the similarity (distance) between clusters. In the single linkage, the similarity between clusters is measured by the similarity between the closest sequences. And in the complete linkage, the similarity is measured by the most dissimilar sequences. The third is the average linkage, it is measured by the average similarity between all pairwise similarities. The average linkage is also known as the unweighted pair group method with arithmetic mean (UPGMA) [7] which is usually used in the construction of phylogenetic guide trees. Due to the different linkage criteria, the time complexity of the hierarchical clustering algorithm is $O(N^2)$ or even worse $O(N^2 logN)$ [8].

But in practice, hierarchical clustering is hard to handle large-scale datasets even on modern computing systems. For example, to deal with millions of sequences, the hierarchical

*Corresponding author

978-1-7281-1867-3/19/$31.00 ©2019 IEEE

clustering method requires several Terabytes of memory to store the similarity matrix - not to mention the computational consumption.

Another popular method is the greedy clustering algorithm which requires sequences sorted by the length in non-increasing order. Then in each iteration, the longest remaining sequence is chosen as a new cluster centroid (representative sequence). After that, a cluster is built by searching through all the unclustered sequences with a certain threshold determined by users. If the similarity between an unclustered sequence and a cluster centroid is greater than the threshold, this unclustered sequence is marked as belonging to the cluster centroid (redundant sequence) and will not be taken into consideration any longer. The greedy method is much more efficient than hierarchical clustering due to the lower computing and space complexity.

There are other sequence clustering algorithms as well, such as CROP [9] and DACE [10]. CROP is an unsupervised Bayesian clustering and DACE is based on a Dirichlet Process Means (DP-Means) algorithm.

The most fundamental step in clustering is the measurement of similarity between sequences. This step is normally referred to as the alignment. The performance of the alignment algorithm directly affects the accuracy and efficiency of clustering analysis. To balance the accuracy and efficiency of pairwise alignments, typically more than one similarity measurement is used during clustering. In many cases, a less accurate but faster (low complexity) alignment method is considered as a filter to remove unnecessary precise alignments. Then a more accurate but slower algorithm is applied to get the similarities. This strategy is used in our implementation as well as CD-HIT, UCLUST, DNACLUST and many other applications.

The rest of this paper is organized as follows: Section II gives a short discussion about parallel clustering implementations in recent years. Then we describe the details of our method in Section III. And we evaluate our distributed framework in Section IV. Finally, we conclude this paper in Section V.

II. Related work

TABLE I
HERE WE LIST THE MAIN CHARACTERS OF POPULAR PARALLEL CLUSTERING TOOLS. CM STANDS FOR CLUSTERING METHOD. AND MN IS MULTI-NODE.

Tool	CM	Alignment		MN
		Filter	Align	
CD-HIT [11]	Greedy	Word table	Semi-global	No
UCLUST [4]	Greedy	USEARCH		No
DNACLUST [5]	Greedy	Kmer	Semi-global	No
HPC-CLUST [12]	Hierarchical	Pre-aligned		Yes
DACE [10]	DP-Means	SA	Banded DP	Yes
SLAD [13]	Landmark	USEARCH		Yes

To exploit the computing power of parallel and distributed computing systems, plenty of parallel methods have been proposed recently (see Table I). CD-HIT was reported to accelerate clustering by implementing a multi-threading database similarity searching in each iteration. Given T threads, the basic idea is to use two word tables and use $T-1$ threads to run multiple checking procedures using one word table (an immutable checking table). And the remaining thread runs a single clustering procedure using the other table (a mutable clustering table) in parallel.

Another widely used tool called UCLUST adopted a similar greedy pattern with CD-HIT, but it used USEARCH to accelerate computing pairwise similarity. Now, UCLUST is a part of the USEARCH project.

For hierarchical clustering, several attempts including HPC-CLUST [12], DACE [10] and SLAD [13] have been proposed to accelerate the clustering process by exploiting the power of parallel computing. However, some of these existing methods including CD-HIT and UCLUST can only work on a single computing node instead of on a distributed computing cluster. The limited resources in a single node constraint the scalability for handling ultra-large-scale databases.

Some tools mentioned above were implemented on distributed clusters. HPC-CLUST took a pre-aligned profile as input, which is rather expensive to obtain. And for DACE, the similarity measurement using locality sensitive hashing should be further discussed. The recent reported SLAD adopted a landmark-based active divisive clustering (LDAC) method to handle ultra-large-scale datasets. In their implementation, a database is first partitioned via LDAC, then sub-clustering operations such as UCLUST were performed via Apache Spark. The performance of SLAD was somehow constrained by UCLUST.

Overall, some of these existing tools cannot extend to distributed clusters. And the others lack the ability to deal with extremely long sequences. To resolve these issues, we've developed a new framework to support large-scale and long sequences.

III. Method

A. Overview

In this paper, we've developed a distributed parallel framework for the greedy clustering method to process ultra-large-scale and extremely long sequences on distributed systems. Fig. 1 gives an overview of this framework. This framework is implemented in multi-level parallelism:

- Level 1 - Node level;
- Level 2 - Parallelism inside a chunk;
- Level 3 - Pairwise alignment level.

Theoretically, multiple similar metrics are supported in our framework. But we choose a SSA-based alignment to compute similarities of sequences by MUMi (Maximal Unique Match indicates) distance [14], considering its better performance for handling long sequences. Due to its low computing complexity, it is much faster than traditional alignment algorithms such as blast [15] and global alignment [16].

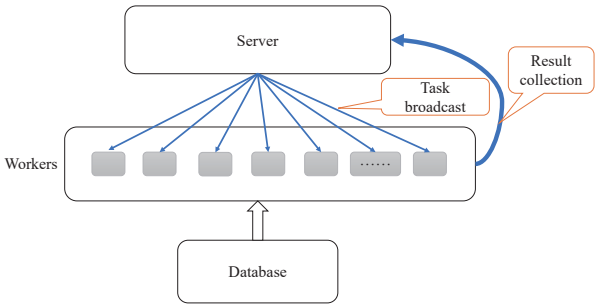

Fig. 1. An overview of our framework.

The procedure of our framework contains four main steps. Step one, the whole input database is sorted and partitioned into chunks to guarantee the load balance for each worker node. Step two, the server node loads a chunk of sequences (reference sequences) and broadcasts the chunk and its corresponding extra-data to each worker node after a series of operations. Step three, each worker node, after receiving reference sequences and the extra-data, calculates pairwise similarities between reference sequences and sub-database before clustering. Repeat step two and step three until all sequences have been processed. Step four, the server node then collects and merges all clustering results from all worker nodes at last.

B. Data partition

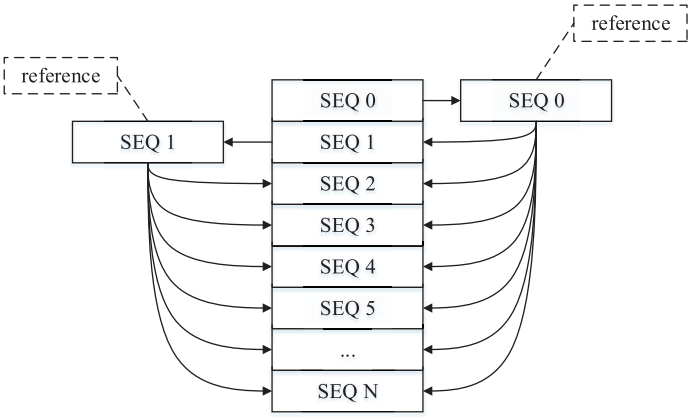

Fig. 2. After pre-sorting, one sequence only need to do pairwise alignments with the sequences below.

As for the greedy method, the very first step is to sort the input database by sequences' length [17]. It's easy to find that for the greedy clustering method one sequence only needs to do the pairwise alignments with the sequences below it (see Fig. 2). So workload will decrease iteration by iteration and this will cause a severe load imbalance problem when the whole database is simply partitioned into chunks as shown in Fig. 3. In Fig. 3, after M iterations, there is no task in worker$_1$ while the load in other worker nodes won't reduce. For distributed parallel computing systems, to achieve better

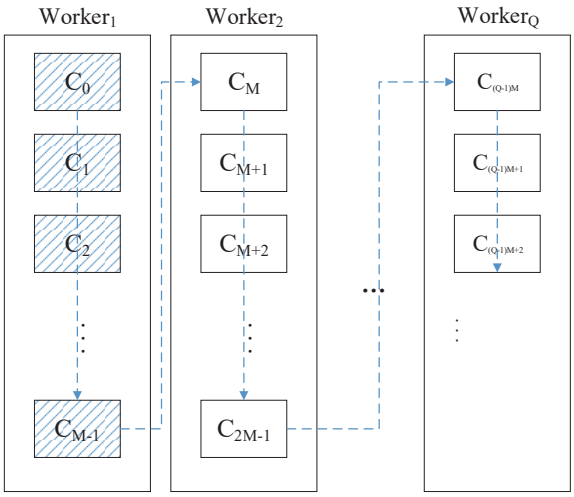

Fig. 3. After pre-sorting, one sequence only needs to do pairwise alignments with the sequences below.

load balance, a proper partition strategy is of vital importance. In our framework, a partition in modulo way is applied to make sure that in each iteration, the computing load of all workers is approximately the same (in a difference of one chunk at most). Our method is shown in Fig. 4.

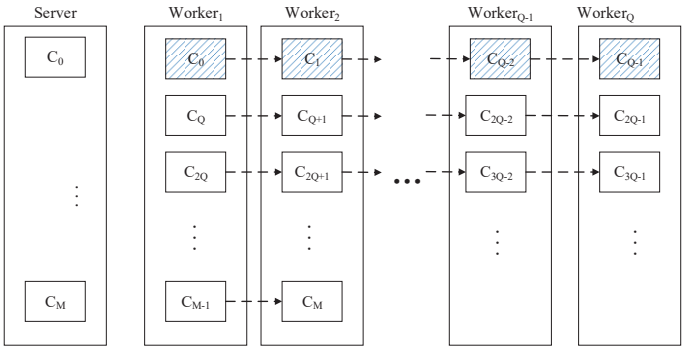

Fig. 4. Database is parted in modulo way. The task of each worker node is approximate.

C. Large-scale inter-node parallelization

We implement a multi-level parallel clustering framework that is specially designed for the greedy method. The basic idea of our method is similar to CD-HIT and UCLUST. There are some new challenges when we implement it on a distributed cluster system. The first problem is the load imbalance of worker nodes in each iteration, and we have introduced our partition strategy above to solve it. The next problem is that there is a heavy overhead of communications among all nodes. This is caused by the consistency of the reference sequences and extra information in every worker node **and the sequentiality of extra information between two adjacent iterations.** For the computation of MUMi distance in our

framework, extra information is essential such as suffix array (SA), inverse suffix array (ISA) and longest common prefix (LCP).The last one, an inner clustering is necessary for each reference chunk to guarantee the correctness. It is also a time-consuming operation. To solve these two problems, we design an efficient strategy of transformation-computation overlaps. A server node is used to prepare reference sequences and extra information, then the server node will do the inner clustering while worker nodes are computing. Fig. 5 illustrates how our framework works in detail.

Here we show the procedure briefly. After initialization, the server node performs an inner clustering against the first chunk C_0 primarily. If any sequence was found belonging to a cluster centroid, the server node will rebuild the related information including SA, ISA, and LCP of the first chunk C_0 and then pack new extra data structures to the MPI broadcast buffer. In the mean-time, worker nodes load sub-database including several chunks from HDD to memory. A barrier is set here to ensure all nodes get ready for the broadcast operation from the server node. After receiving reference sequences (C_0) and necessary extra information, worker nodes will do the similarity search between reference sequences and sequences of sub-database. While worker nodes are searching sub-database, server nodes are doing inner clustering and packing data for the next iteration. To avoid unnecessary computation, when preparing extra data for the next iteration, the server node will synchronize simplified clustering information of this chunk with the corresponding worker. If a sequence is marked as clustered it will be ignored for the rest of iterations. When the search of chunk 1 against chunk 0 is finished, the clustering information will be synchronized to the server node immediately. This method is illustrated in Fig. 5. And then we do the same operations to remaining chunks.

1) Transformation-computation overlaps: A key improvement in our framework is that the overhead of computations among server node and worker nodes can be eliminated by overlapping communication and computation (see Fig. 6). This overlapping strategy is based on the fact that reference sequences and their clustering information can be obtained at the start of the searching process instead of at the end of searching. So the server node can prepare and broadcast-related information for the next reference chunk while worker nodes are searching against the sub-database. As a result, except for the first chunk, the overhead of communication can be eliminated using fine-tuned chunk size.

2) Node level parallelism: We use MPI to control the behavior of server and worker nodes to make them collaborate in the node-level parallelism. A barrier is essential to ensure the accuracy of our framework. Besides, a broadcast operation is necessary when synchronizing reference sequences for each worker node. To achieve better-communicating efficiency, the clustering related data is encoded to a single buffer. So our framework only needs to perform the broadcast operation once per iteration. Our methods make sure that worker nodes do clustering to different chunks in a parallel way. The rest two-level parallelization will be shown below.

D. *Intra-node parallelization*

1) Clustering: The clustering of one chunk is performed in a single node with a dynamic multi-threading greedy approach. The pseudo-code is shown in Algorithm 1. It follows the basic pipeline of CD-HIT mentioned above.

Algorithm 1 Clustering Algorithm

```
Input:
Chunk - need to be clustered
SA - SA of Chunk_ref (reference chunk)

procedure CLUSTERING(Chunk,SA)
    for each seq_i in Chunk do
        Collect Maximal exact matches (Mems) of seq_i against SA
        Get sum of all length of Mems
        if sum > Filterthreshold then
            ComputeMemIdentity of seq_i
        end if
    end for
end procedure

Output:
Totalgenomes - simplified information of clustering result after updating with part result of this inner cluster
```

However, there is a little difference in implementation between the server and workers. The Chunk and Chunk_{ref} are the same one in the server node while they are different in worker nodes. Also, for seq_k of *Chunk*, all sequences of $\textit{Chunk}_{ref}$ will be searched in workers. In comparison, we only search the sequences longer than seq_k in $\textit{Chunk}_{ref}$. The clustering works on the server node, it's much less powerful compared to worker nodes. So chunk size needs to be carefully chosen to balance the computing load on both the server node and work nodes. When searching the database, normally not all alignments are necessary under a user-specified threshold. So practically a filter based strategy is used to reduce unnecessary sequence comparisons (see Fig. 7). With a high threshold, 90% is set in default in our implementation, most of the time-consuming alignments can be avoided.

2) Parallelization: The clustering on a single node is a time consuming operation, we implement two levels of parallelism to accelerate this step. In the chunk level, part of the sub-database is loaded into a pre-allocated I/O buffer. Sequences inside a chunk are independent of each other. So all threads work together to separately find whether the sequences belonging to the reference chunk. For inner alignment parallelism, it is usually used in two extreme circumstances. One is that when only several pairs of sequences alignments left (see Fig. 8 (a)). The other is there are extremely long sequences and others are short sequences (see Fig. 8 (b)). In these cases, only minor threads are working. To exploit all the computing power, we introduce an inner alignment parallelism, for K-way sparse SA. It can process one sequence with K threads.

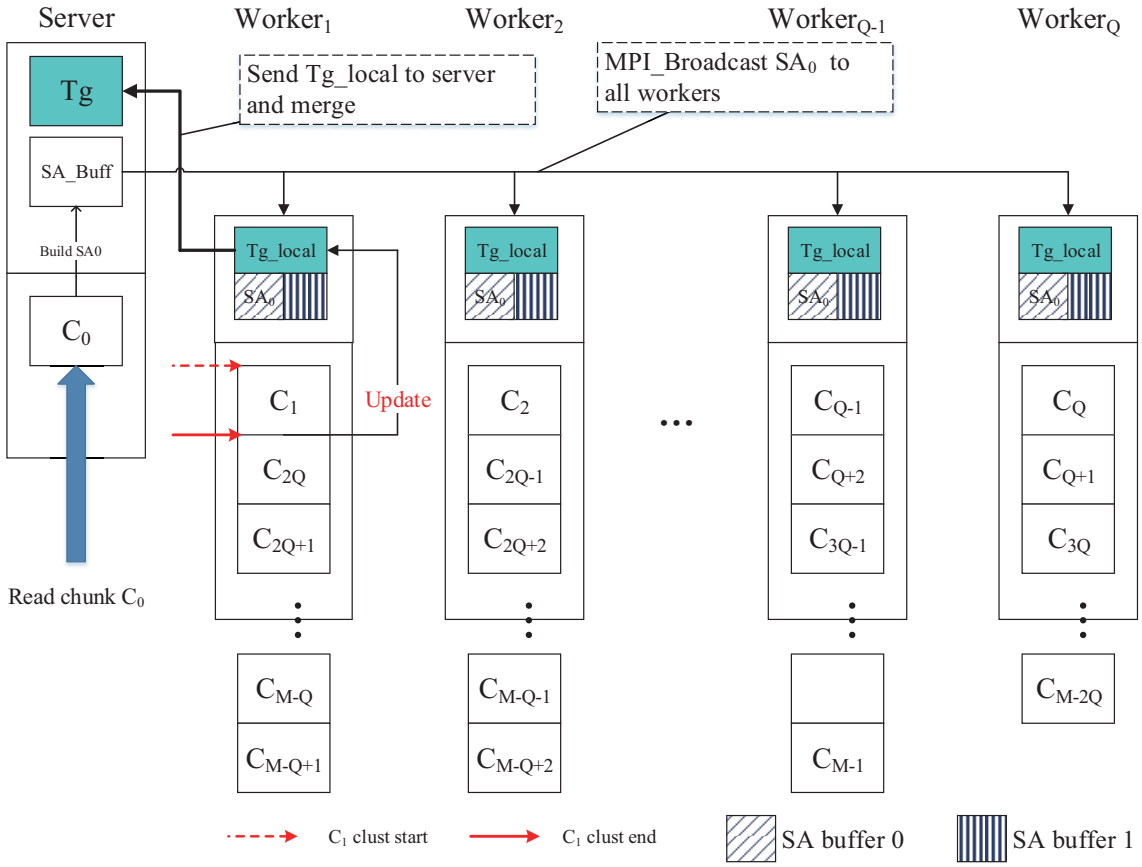

Fig. 5. Tg is cluster information in server node. Tg_local means a part of cluster information in each worker node. And for each worker node, two SA_buffer is needed to store SA for current round and next round.

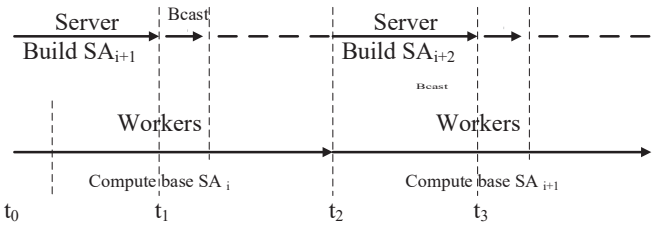

Fig. 6. Pipeline of server and worker nodes. Computation and transformation are overlapped.

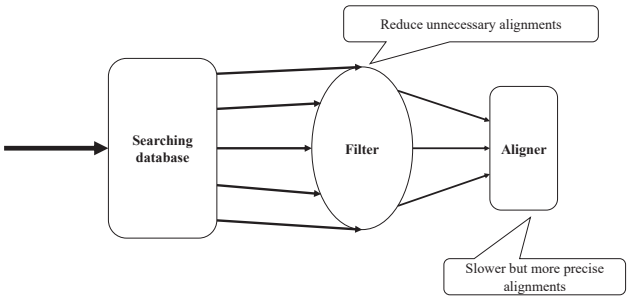

Fig. 7. Filter strategy is used to reduce unnecessary alignments under a user specified threshold.

3) Computing pairwise sequence similarity: Our pairwise alignment is based on sparse SA [18] and MUMi [14]. Sparse SA is used to find MEMs we mentioned in Algorithm 1. It occupies much less memory in contrast to a full-text index. Because Sparse SA only stores every K-th position of the sequence, which is different from a full index storing all positions. MUMi is a genomic-distance index based on maximal unique matches (MUM) shared by two genomes. After finding all MEMs, we can get the MUMi by removing MEM overlaps. This alignment pipeline is more suitable for long sequences with a quasilinear complexity comparing to those global or local alignment algorithms. Due to the low complexity, this method is rather efficient for dealing with extreme long sequences without huge memory consumption. And we have done following enhancements to the original sparse SA implementation:

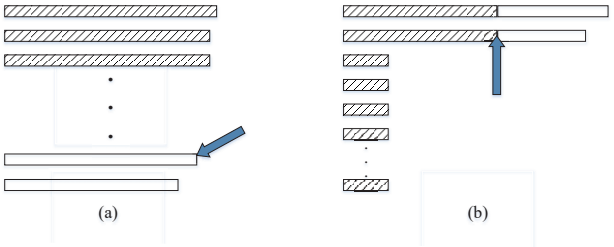

Fig. 8. (a) Only two unaligned sequences are left on a worker node. (b) Only the alignment of the first two extremely long sequences is still unfinished.

- We change the static multi-threading schedule approach to a dynamic approach.
- A double buffer strategy is used when reading the database from HDD to memory (see Fig. 9).

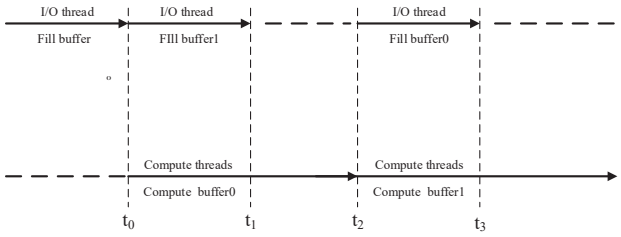

Fig. 9. Double buffering pipeline.

These enhancements make the pairwise alignments more efficient and scalable. In the original implementation, it adopts a static instead of dynamic multi-threading approach. But, we get a performance improvement by using a dynamic approach to minimize the overhead of thread synchronizations. And our double-buffering strategy makes a single node able to handle larger databases even beyond the capacity of memory without any extra overhead.

For different kinds of research purposes, particular alignment algorithms may be required. Or researchers may want to compare the difference of clustering results using multiple alignment algorithms. So it's meaningful to support more than one alignment algorithm in this framework. In our implementation, we've reserved interfaces to call other alignment algorithms by linking other well-optimized alignment libraries such as [11], [19]–[21], even for alignment-free method mentioned in [22].

IV. Performance Evaluation

We have performed experiments on large-scale datasets with extreme long sequences from the NCBI RefSeq database. The results demonstrate that our framework achieves linear speedups for the worker node number. Especially, our framework supports extreme long sequences up to millions of bps.

A. Evaluation Environment

The performance evaluation on a single node is done on a 12 cores server with double E5-2620 v2 CPUs, see the first row of Table II. And we have evaluated the scalability of our framework on a distributed memory cluster with 10 nodes (200 cores), configurations are shown in the second row of Table II.

TABLE II
System configurations of evaluation environment.

CPU	Memory	Interconnection
E5-2620 v2 @2.1Hz (6C) * 2	16GB DDR3	None
E5-2680 v2 @2.8Hz (10C) * 2	64GB DDR3	56Gb FDR InfiniBand

B. Datasets and parameters

TABLE III
The detailed information of datasets.

Dataset	Size(Mbp)	MaxLen(bp)	MinLen(bp)	AveLen(bp)
Viral	261	2,473,870	200	23,290
Archara	2,028	5,751,492	22	52,845

We've chosen datasets from the NCBI RefSeq database to test the performance of our framework. Viral and Archaea are downloaded from ftp://ftp.ncbi.nlm.nih.gov/refseq/release/. Detail information of these two datasets is shown in Table III. The longest sequence is more than five million bps, which is a severe challenge for other clustering methods.

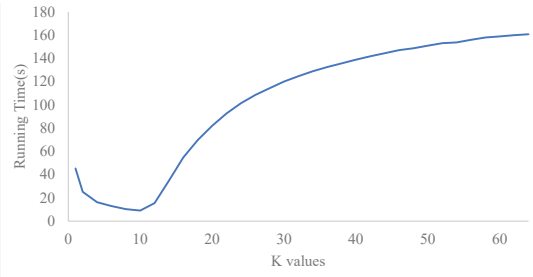

Fig. 10. The alignment performance with different Ks on a subset of Viral.

TABLE IV
Main parameters of our framework.

Parameter	Threads	K	Threshold	MIN_LEN
Value	Cores	10	0.9	20

The parameters of our experiments are listed in Table IV. In our experiments, the similarity threshold is set to 0.9, the thread number in each worker node is equal to the number of cores. On the E5-2680 v2 based node, 20 threads are used to do the pairwise alignments. Besides, the sparse granularity is 10 ($K = 10$) for our SSA-based alignment method. We find that the value of K has a strong influence on alignment performance (see Fig. 10). For our experiments, the length of minimal MEM is 20 due to the testing dataset contains very long sequences.

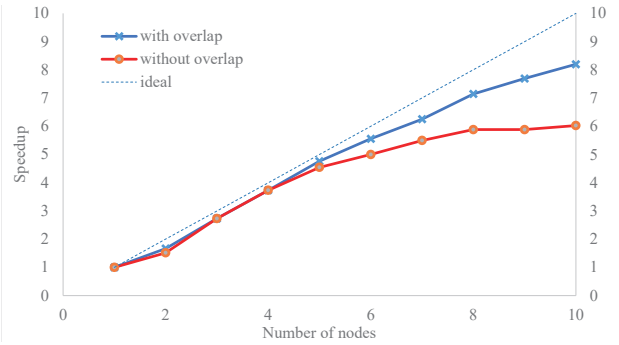

Fig. 11. Scaling evaluation of overlap vs non-overlap.

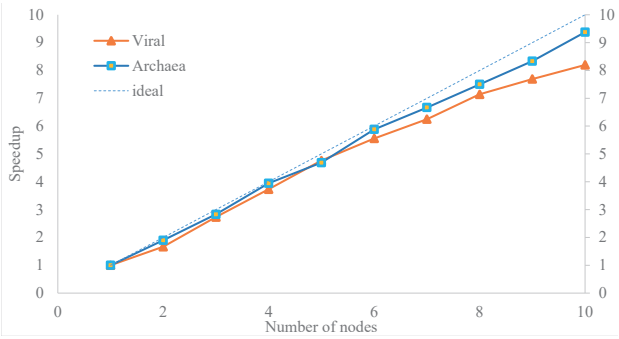

Fig. 12. Scaling performance framework.

C. Evaluations

1) Framework scaling evaluation: We have performed a strong scaling test using two versions of our framework: with/without communication-computation overlap on the Viral dataset. Then a larger dataset Archaea is tested using the overlapping version. We find that for the small-scale scenario, both versions scale well. It seems that the overhead of communications is tolerable when using less than 5 nodes. But, when more nodes are used, the overlapping strategy makes a great performance improvement (see Fig. 11). And Fig. 12 illustrates that this framework achieves near-linear speedups for the node number. And it's obvious to find that DGCF performs better when dealing with the larger dataset Archaea comparing with Viral. This is because the Viral dataset is too small that only a small number of chunks residing in each worker node.

2) Node performance scaling: Furthermore, we test how our alignment method scales when adapting a dynamic multi-threading method using a Viral dataset using a single node. Fig. 13 shows that this SSA-based method achieves linear speedups for the thread number. On a 20-core node, our multi-threading algorithm implementation achieves more than 18x speedup comparing with a single thread.

3) Comparison with other methods: Comparing with other distributed clustering methods such as DACE or SLAD, the accuracy of our framework is easy to be verified because we use the well studied greedy method adopted by CD-HIT and UCLUST. We have planned to compare our performance with DACE and SLAD, but unfortunately, DACE cannot support the

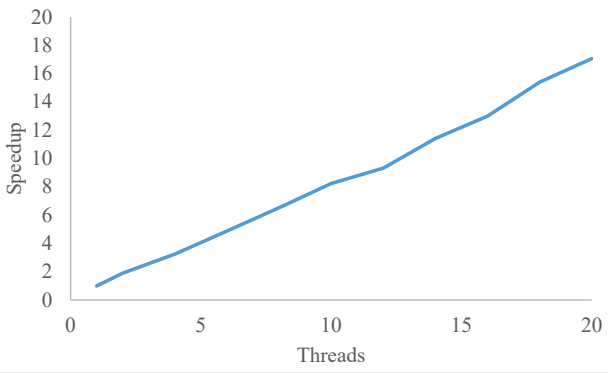

Fig. 13. Scaling performance on single node.

TABLE V
THE RUNNING TIME (SECOND) OF DIFFERENT ALIGNMENT METHOD USING SIMULATED SEQUENCES IN DIFFERENT LENGTH.

	1k	10k	100k	500k	1M	3M
Our method	0.008	0.06	0.85	5.87	10.8	37.3
CD-HIT	0.16	0.17	0.27	0.66	88.5	1678
USEARCH/UCLUST	0.004	0.094	No	No	No	No
ESPRIT/GLOBAL	0.009	25.74	>8h	No	No	No

Viral dataset. And USEARCH (the kernel of SLAD) cannot support sequences longer than 500 thousand bps. Thus, we test several widely used alignment implementation in clustering tools to find their performance for handling long sequences. The results are shown in Table V. It seems that most of them don't support sequences longer than 500 thousand bps except for our method and the word table method in CD-HIT. And when dealing with long sequences, for example, the 3 million bps sequence, our method is about 50 times faster than CD-HIT. our method outperforms any other alignment methods when meeting extremely long sequences.

V. Conclusion

Clustering is one of the most fundamental steps in sequence analysis. New sequencing technologies such as NGS and 3GS have dramatically increased both the dataset size and the length of a single sequence. However, existing tools suffer from the worse scalability for handling the large-scale datasets as well as the long sequences. With the development of high-performance distributed computing systems, it's of vital importance to design scalable clustering methods to overcome these bottlenecks. In this paper, we have introduced a distributed clustering framework that is specially designed for handling large-scale datasets and extremely long sequences. Our framework adopts a greedy clustering strategy and enables the overlap of communication and computation between distributed computing nodes. To deal with extremely long sequences, a SSA-based algorithm is used. Our framework is implemented using MPI and OpenMP hybrid programming model. Experiments show that our framework achieves near-linear speedups and can handle long sequences efficiently.

VI. ACKNOWLEDGEMENTS

This work is partially supported by the National Key R&D Program of China (Grant No. 2018YFB0203903); the National Natural Science Foundation of China (Grant No. U1806205 and 31771466); the CAS 100-Talents (Dr. Beifang Niu); the Shenzhen Basic Research Fund (Grant No. JCYJ20180507182818013); Center for High Performance Computing and System Simulation, Pilot National Laboratory for Marine Science and Technology (Qingdao).

REFERENCES

[1] Y. Sun, Y. Cai, L. Liu, F. Yu, M. L. Farrell, W. McKendree, and W. Farmerie, "Esprit: estimating species richness using large collections of 16s rrna pyrosequences," *Nucleic acids research*, vol. 37, no. 10, pp. e76–e76, 2009.

[2] W. Li, L. Jaroszewski, and A. Godzik, "Clustering of highly homologous sequences to reduce the size of large protein databases," *Bioinformatics*, vol. 17, no. 3, pp. 282–283, 2001.

[3] W. Li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences," *Bioinformatics*, vol. 22, no. 13, pp. 1658–1659, 2006.

[4] R. C. Edgar, "Search and clustering orders of magnitude faster than blast," *Bioinformatics*, vol. 26, no. 19, pp. 2460–2461, 2010.

[5] M. Ghodsi, B. Liu, and M. Pop, "Dnaclust: accurate and efficient clustering of phylogenetic marker genes," *BMC bioinformatics*, vol. 12, no. 1, p. 271, 2011.

[6] M. Steinegger and J. Söding, "Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets," *Nature biotechnology*, vol. 35, no. 11, p. 1026, 2017.

[7] I. Gronau and S. Moran, "Optimal implementations of upgma and other common clustering algorithms," *Information Processing Letters*, vol. 104, no. 6, pp. 205–210, 2007.

[8] W. H. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of classification*, vol. 1, no. 1, pp. 7–24, 1984.

[9] X. Hao, R. Jiang, and T. Chen, "Clustering 16s rrna for otu prediction: a method of unsupervised bayesian clustering," *Bioinformatics*, vol. 27, no. 5, pp. 611–618, 2011.

[10] L. Jiang, Y. Dong, N. Chen, and T. Chen, "Dace: a scalable dp-means algorithm for clustering extremely large sequence data," *Bioinformatics*, vol. 33, no. 6, pp. 834–842, 2016.

[11] L. Fu, B. Niu, Z. Zhu, S. Wu, and W. Li, "Cd-hit: accelerated for clustering the next-generation sequencing data," *Bioinformatics*, vol. 28, no. 23, pp. 3150–3152, 2012.

[12] J. F. Matias Rodrigues and C. von Mering, "Hpc-clust: distributed hierarchical clustering for large sets of nucleotide sequences," *Bioinformatics*, vol. 30, no. 2, pp. 287–288, 2013.

[13] W. Zheng, Q. Mao, R. J. Genco, J. Wactawski-Wende, M. Buck, Y. Cai, Y. Sun, and I. Birol, "A parallel computational framework for ultra-large-scale sequence clustering analysis." *Bioinformatics (Oxford, England)*, 2018.

[14] M. Deloger, M. El Karoui, and M.-A. Petit, "A genomic distance based on mum indicates discontinuity between most bacterial species and genera," *Journal of bacteriology*, vol. 191, no. 1, pp. 91–99, 2009.

[15] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[16] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[17] U. Hobohm, M. Scharf, R. Schneider, and C. Sander, "Selection of representative protein data sets," *Protein Science*, vol. 1, no. 3, pp. 409–417, 1992.

[18] Z. Khan, J. S. Bloom, L. Kruglyak, and M. Singh, "A practical algorithm for finding maximal exact matches in large sequence datasets using sparse suffix arrays," *Bioinformatics*, vol. 25, no. 13, pp. 1609–1616, 2009.

[19] H. Lan, W. Liu, B. Schmidt, and B. Wang, "Accelerating large-scale biological database search on xeon phi-based neo-heterogeneous architectures," in *Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 503–510.

[20] C. NVIDIA, "Zone: nvbio."

[21] A. Döring, D. Weese, T. Rausch, and K. Reinert, "Seqan an efficient, generic c++ library for sequence analysis," *BMC bioinformatics*, vol. 9, no. 1, p. 11, 2008.

[22] S. Vinga and J. Almeida, "Alignment-free sequence comparisona review," *Bioinformatics*, vol. 19, no. 4, pp. 513–523, 2003.