A MapReduce Framework for Mining Maximal Contiguous Frequent Patterns in Large DNA Sequence Datasets

Md. Rezaul Karim, Md. Azam Hossain, Md. Mamunur Rashid, Byeong-Soo Jeong and Ho-Jin Choi[1]

Department of Computer Engineering, Kyung Hee University, [1]Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea

Abstract

Current DNA sequence datasets have become extremely large, making it a great challenge for single-processor and main-memory-based computing systems to mine interesting patterns. Such limited hardware resources make the performance of most Apriori-like algorithms inefficient. However, recent implementation of a MapReduce framework has overcome these limitations. Furthermore, mining with maximal contiguous frequent patterns to express the function and structure of DNA sequences is a useful technique, capable of capturing the common data characteristics among related sequences. In this paper, we proposed an efficient approach for mining maximal contiguous frequent patterns in large DNA sequence data using MapReduce framework which can handle a massive DNA sequence datasets with a large number of nodes on a Hadoop platform. Our extensive experimental results show that the proposed approach can mine the complete set of maximal contiguous frequent patterns very efficiently.

Keywords

DNA sequence datasets, Hadoop, MapReduce, Maximal contiguous frequent patterns.

1. Introduction

In living organisms, DNA does not usually exist as a single molecule, but instead as a pair of molecules that are held tightly together, so one of the most interesting challenges is to discover sequences that are similar or identical between different genomic locations or different genomes.

Similar sequences may be present because they have been conserved or selected during evolution due to some mediating important biological functions. Because of the size of the DNA sequence, data are very large, and the character set is small; the short patterns are almost frequent without exception. From them, biologists assemble whole genome of species based on frequent contiguous sequences.

How to efficiently discover long frequent patterns poses a great challenge for existing sequential pattern mining algorithms. The reason of the maximal contiguous frequent pattern mining is that it is also the frequent pattern, the boundary of frequent pattern set.

Therefore, in this paper we proposed a MapReduce and Hadoop based technique for mining maximal contiguous frequent patterns in a large DNA sequence dataset for the first times ever. This paper is organized as follows. Section II surveys related works. Section III represents the problem regarding the maximal contiguous frequent pattern mining. Section IV represents our proposed MapReduce framework for mining maximal contiguous frequent patterns and The MCFP algorithm. Section V represents experimental results. And finally, we conclude our paper at section VI.

In this paper, we used the term "sub-sequences" and "patterns"; "database" and "datasets"; "concatenated", and "contiguous" interchangeably. And, we also used the acronym PDB for Projected Database; BPs for Base Pairs; *CFP* for Contiguous Frequent Patterns, and *MCFP* for Maximal Contiguous Frequent Patterns.

2. Related Works and Background Study

Many researches have been done in the field of biological sequence mining. The problem in finding the maximal contiguous frequent pattern is of substantial importance to bioinformatics and is widely examined in the literature [1-3]. Based on the idea of Apriori [4], a more efficient algorithm called PrefixSpan [5] has been proposed in recent years. Its general idea is to examine only the prefix sub-sequences and project only their corresponding postfix sub-sequences into PDBs. In each PDB, sequential

sequences are grown by exploring local length-1 frequent sequences. Moreover, a main-memory-based pseudo-projection technique was developed to save the cost of projection when the PDB and its associated pseudo-projection processing structure could be fit in the main memory. However, when mining long frequent concatenated sequences, this method is inefficient. Therefore, it is impractical to apply PrefixSpan [5] to mine long contiguous sub-sequences from biological sequence datasets.

Pan *et al*. [6] first introduced the concept of the variable length spanning tree method to mine maximal concatenated frequent sub-sequences. They, proposed two algorithms called MacosFSpan and MacosVSpan based on PrefixSpan [5], that effectively reduces the recursive process. Although the MacosVSpan algorithm is very efficient for mining long concatenated frequent sub-sequences, the MacosFSpan have some limitations making it unsuitable for mining long contiguous frequent patterns; first, it constructs length-4 fixed length sub-sequence candidates; then, candidates of length-5, length-6, etc. It is very time-consuming process. Second, it did not consider about the size of PDBs and reality is that the size of physical PDBs are very large compared to the original datasets and in most of the cases, these PDBs will not fit into the main memory. Third, fixed length scanning method is very inefficient for large sequence datasets, because of millions of BPs. Fourth problem is that both MacosVSpan and MacosFSpan uses in-memory pseudo PDBs using pointer-offset pairs only; but only pointer-offset pair are not enough for processing all the prefix and suffixes.

In literature [7], the authors claimed that their proposed algorithm is more efficient than MacosVS algorithm; but for mining long frequent concatenated sequence, MacosVS algorithm is much more efficient than MacosFSpan algorithm and basically literature [7] was proposed based on [6] and needs multiple times database scanning. Although this approach reduces the recursive execution process for expanding sub-sequences, but it also has the problem of creating and processing with projected databases. Thus, this approach is also inefficient to produce the result in a faster way.

Therefore, from the above surveys, it is clear that the traditional contiguous frequent pattern mining algorithms [5-7] usually takes a DNA sequence database and generates candidate item sets using fixed length spanning tree or suffix tree-based algorithm [1,2,8]. Surprisingly, all of the previous works [1,2,4-9] assumed that the DNA sequence datasets, associated tree structure, and all the associated projected databases could be fit into the main memory, but many practical DNA sequence database size is huge, sometimes 100 GBs. So, certainly, these will not fit into the main memory.

On the other hand, parallel and partitioning approaches in distributed environments also been introduced to mine frequent patterns from sequence databases; however, in distributed environments, communication cost is huge because of many message passing, data sharing, and I/O operations. These poses very impractical to use distributed systems for mining large datasets. That is why MapReduce [10-12] is a very suitable framework to mine these sorts of datasets, where it only needed to share and pass the support of individual candidate item set rather passing the candidate item set itself. Therefore, definitely communication cost is very low compared to the distributed environments.

2.1 MapReduce

MapReduce was developed within Google [9] as a mechanism for processing large amounts of raw data, for example, crawled documents or web request logs. These data are so large; it must be distributed across thousands of machines in order to be processed in a reasonable time. This distribution implies parallel computing since the same computations are performed on each CPU, but with a different dataset. The user of the MapReduce library expresses the computation as two functions: Map and Reduce [10-12]. It merges together these values to form a possibly smaller set of values. Typically, just zero or one output value is produced per reduce invocation. The intermediate values are supplied to the user's reduce function via iterator. This allows us to handle lists of values that are too large to fit in the main memory. MapReduce provides an abstraction that involves the programmer defining a "mapper" and a "reducer," with the following signatures [12]:

- Map: (value 1, key1) → list (key2, value2)
- Reduce: (key2, list (value2) → list (value2).

2.2 Hadoop and the Hadoop Distributed File System

Hadoop is a popular open source implementation of MapReduce, which is a powerful tool designed for deep analysis and transformation of very large datasets which is inspired by Google's MapReduce and Google File System [10]. It enables applications to work with thousands of nodes and petabytes of data.

Hadoop uses a distributed file system called Hadoop Distributed File System (HDFS) [13], which creates multiple replicas of data blocks and distributes them on computer nodes throughout a cluster to enable reliability and has extremely rapid computations to store data as well as the intermediate results [14]. The Hadoop runtime system coupled with HDFS manages the details of parallelism and concurrency to provide ease of parallel programming with reinforced reliability. In a Hadoop cluster, a master node controls

a group of slave nodes on which the Map and Reduce functions run in parallel.

3. Problem Statement

In this section, first we define the problem of maximal contiguous frequent pattern mining and then present some preliminary knowledge that will be used in our algorithm.

Let $\sum$ = {A, C, G, T} be a set of DNA alphabets where A, C, G, and T are called DNA characters or four bases; A is for *Adenine*, C for *Cytosine*, G for *Guanine,* and T for *Thiamine*. A DNA sequence *S* is an ordered list of DNA alphabets. *S* is denoted by $<s_1, s_2 \ldots s_n>$ where $s_i \in \sum$ and $|S|$ denotes the length of sequence *S*. A sequence with length *n* is called an *n*-sequence.

A DNA sequence database *D* is a set of tuples <*Sid, S*> where *Sid* is a sequence identifier and *S* is the corresponding sequence. A sequence $\alpha = <a_1,a_2,\ldots,a_n>$ is called contiguous sub-sequence of another sequence $\beta = <b_1,b_2,\ldots,b_m>$ and β is a contiguous super-sequence of α, denoted as $\alpha \subseteq \beta$, if there exists integers $1 \le j_1 \le j_2 \le \ldots \le j_n \le m$ and $j_{i+1}=j_i+1$ for $1 \le I \le n-1$, such that $a_1=b_{j1}$, $a_2=b_{j2},\ldots,a_n=b_{jn}$. We can also say that α is *contained* by β. A contiguous frequent sub-sequence *X* is said to be maximal if none of its super-sequence *Y* is frequent.

Given a DNA sequence database *D* and a minimum support threshold δ, the problem of maximal contiguous frequent sub-sequences mining is to find the complete set of maximal contiguous frequent patterns from that database.

For example, suppose minimum support threshold δ *is 2* for DNA sequence database in Table 1. Sequence <ATCGTGACT> is 9-sequence since its length is 9. Sequence <ATCG> is contiguous frequent sub-sequence because it is contained by sequences ID 10, 20, and 30. S=<CGTGATT> is a contiguous frequent sub-sequence of length 7 since both sequence identifier 40 and 50 contains it. Moreover, it is one of the maximal contiguous frequent sub-sequences because there is no contiguous frequent super-sequence of <CGTGATT> with minimum support 2.

4. The Proposed MapReduce Framework

4.1 Programming Model

Two important functional programming primitives in MapReduce are Map and Reduce. The Map function is applied on application-specific input data to generate a list of intermediate <key, value> pairs. Then, the Reduce function is applied to the set of intermediate pairs with the same key. The master node assigns a task to a slave node that has any empty task slot. Typically, computing nodes and storage nodes in a Hadoop cluster are identical from the hardware's perspective [15]. Such a homogeneous configuration of Hadoop allows the MapReduce framework to effectively schedule computing tasks on an array of storage nodes where data file are residing, leading to a high aggregate bandwidth across the entire Hadoop cluster.

An input file passed to Map functions resides on the HDFS on a cluster. After that, HDFS splits the input file into even-sized fragments automatically, which are distributed to a pool of slaves for further MapReduce processing.

4.2 Proposed Framework

We know that DNA sequence datasets are usually very large and the number of items is relatively smaller than that of transactional databases. Since every item (Nucleotide) is frequent, there is nothing to mean by 1-itemset; hence, we cannot use MapReduce framework directly for mining these kind of datasets as we can on transactional databases. Therefore, to deal with DNA sequence datasets with MapReduce on Hadoop platform, we need special care like handling big data. DNA sequence datasets in disk files are splitted into smaller segments automatically after they are stored on HDFS. The Hadoop components perform job execution, file staging, and workflow information storage and use the files replace the database to store datasets [13,14].

So, after splitting the DNA datasets into smaller segments, the master node assigns task to idle worker nodes. Table 2 has shown the input/output schemes for the proposed framework. After that, assigned worker nodes scan the sequences in smaller data segments as <*ID, Sequence*> pairs and produce <A>, <T>, <C>, and

Table 1: A DNA sequence database

Sid	Sequence
10	ATCGTGACT
20	CATCGATTG
30	CATCGTGAGA
40	TCGTGATTG
50	GCGTGATTACT
60	AGTCGATTG

Table 2: Key/value for the proposed MapReduce framework

I/O	Map-1	Map-2	Reduce-1	Reduce-2
Input: Key/value pairs	Key: *Sid* Value: *DNA Sequence*	Key: *Serial* Value: *Suffix*	Key: *CP* Value: *Support*	Key: CFP Value: *Support*
Output: Key/value pairs	Key: *Serial* Value: *Suffix*	Key: *Support* Value: *CP*	Key: CFP; Value: *Support*	Key: *MCFP* Value: *Support*

<G> suffixes with serial number (i.e., suffixes starting with four nucleotide A, T, C, and G) as <serial, suffix> pairs. This phase is considered as map phase 1. For example, for a DNA sequence CTGACT, a worker produces five contiguous suffixes with prefix <A>, <T>, <C>, and <G> as: ACT, TGACT, CTGACT, CT, and GACT and will be written in the local disk using serial number as <serial, suffix> pairs. These values are inputted to the map phase 2. For our ease, we designed the map function such that it takes input as <serial, suffix> pairs and perform prefix matching among <A> suffixes, then <T> suffixes, <C> suffixes, and finally <G> suffixes. For example, if we perform the prefix matching for <A> suffixes, then suffix ATTG will match up to ATTG of suffix ATTGCT; so, the map function will produce three new candidate contiguous suffixes as <ATTG, 2> and <ATTGCT,1> pairs, since the support of ATTG will be 2. After having all the required suffixes, the map function executes on these suffix sequences again and generates contiguous candidate suffixes and written as <candi_pattern, support> pairs. This is the end of map phase 2. These <candi_pattern, support> pairs will be treated as intermediate values, where candi_pattern indicates a contiguous candidate suffix.

The reduce function adds up all the intermediate values and produce a support for candidate contiguous suffixes as a one-time synchronization by adding local supports. In the reduce phase, each worker needs extra work for finding maximal contiguous frequent patterns and it is treated as second reduce phase. After having all the contiguous frequent patterns by sharing supports of different worker nodes as <freq_pattern, support> pairs, each worker just checks the maximality criteria among the contiguous frequent patterns and writes maximal contiguous frequent patterns on the output files as <max_pattern, support> pairs.

Actually, in map phase and reduce phase, this algorithm's advantage is that it does not exchange data between data nodes, it only exchanges the supports. Figure 1 describes the algorithm of our proposed approach and Figure 2 shows the data flow and the phases of the

MCFP Algorithm on Hadoop MapReduce

Input: A DNA sequence database on HDFS and a minimum support threshold δ
Output: The complete set of maximal contiguous frequent patterns
Map Phase: Assigned worker nodes scan the splitted segments and maps the output as <candi_pattern, support> pairs
Map Phase-1:
1. Generate <A>, <T>, <C>, and <G> suffixes and write the invoked values on the local disk as <serial, suffix> pairs

Map Phase-2:
2. Worker nodes take input as <serial, suffix> pairs and maps these values as <candi_pattern, support> pairs.
3. Write the <candi_pattern, support> pairs on the local disks

Reduce Phase: Assigned worker nodes find the complete set of MCFPs as <max_pattern, support> pairs
Reduce Phase-1:
1. Worker nodes take input as <candi_pattern, support> pairs and share the support of each candidate suffix with other workers and find the set of CFP as <freq_pattern, support>pairs

Reduce Phase-2:
2. Output from the Reduce phase-1 is inputted to idle worker nodes to check the maximality criteria and write these MCFPs as <max_pattern, support> pairs in the output files

Figure 1: MCFP algorithm on Hadoop using MapReduce.

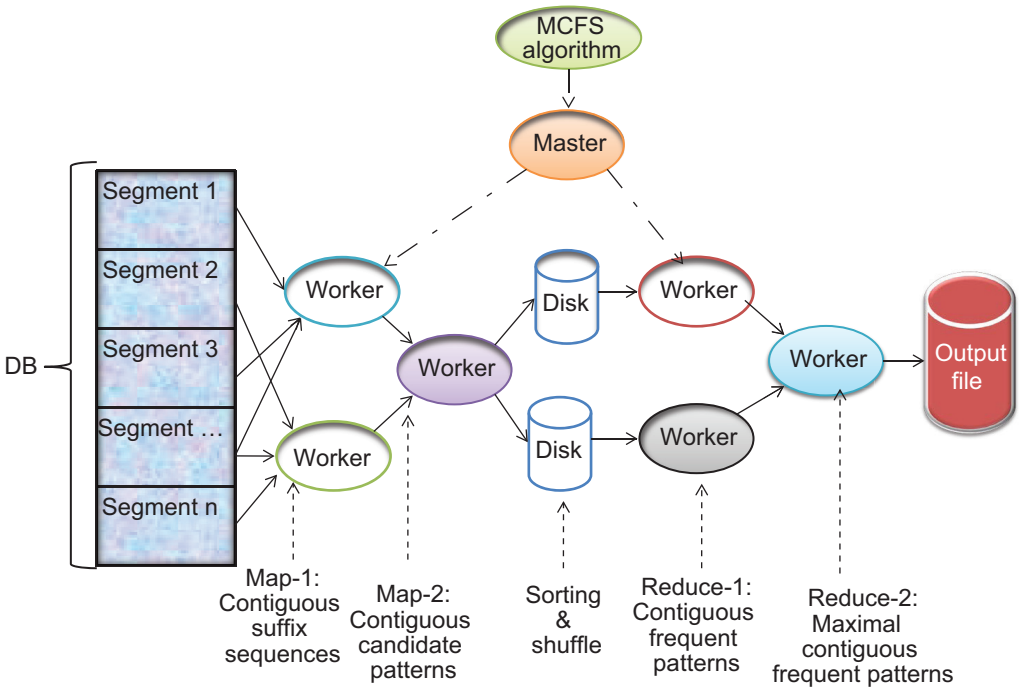

Figure 2: Proposed MapReduce framework for mining maximal contiguous frequent patterns.

MapReduce framework for mining maximal contiguous frequent patterns. Multiple iterations of MapReduce computations are necessary for the overall computation. The iteration continues until there are not any maximal frequent item sets further found.

4.3 Step-by-step Example

4.3.1 Map Phase

i. Suppose the minimum support threshold is 2 and the DNA sequence database are in Table 1 has been splitted into two segments with each three sequences; sequence 10, 20, and 30 in first segment and sequence 40, 50, and 60 are in the second segment. Let, worker nodes 1, 2, 3, 4, 5 and 6 are idle and the master node assigned segment 01 to 1st worker and segment 02 to 2nd worker. Tables 3 and 4 have shown the suffix sequences with serial number using a notation like 1-5 instead of 1, 2, 3, 4, and 5; for the page limitations. These <serial, suffix> pairs are stored on the local disk of worker 1 and 2. These values will be used as input to the map function in map phase 2
ii. Now, in Map phase-2, <serial, suffix> pairs are inputted to the mapping function. Then, the map function runs the prefix matching among the related suffixes and produce results as <candi_pattern, support> pairs. Tables 5 and 6 have shown the <candi_pattern, support> pairs from two workers.

Table 3: Output suffixes from worker 1 in map phase-I

Serial	Suffixes
1-10	ATCGTGACT, ACT, ATCGATTG, ATTG, ATCGTGAGA, AGA, ATTACT, AGTCGATTG
11-17	TCGTGACT, TGACT, TCGATTG, TTG, TG, TCGTGAGA, TGAGA
18-23	CGTCACT, CT, CATCGATTG, CGATTG, CATCGTGAGA, CGTGAGA
24-29	GTGACT, GACT, GATTG, GTGAGA, GAGA, GA

Table 4: Output suffixes from worker 2 in map phase-I

Serial	Suffixes
1-5	ATTG, ATTACT, ACT, AGTCGATTG, ATTG
6-15	TCGTGATTG, TGATTG, TTG,TG, TGATTACT, TTACT, TACT, TCGATTG, TTG, TG
16-19	CGTGATTG, CGTGATTACT, CT, CGATTG
20-23	GTGATTG, GATTG, GCGTGATTACT, GATTACT

Table 5: Intermediate values – From worker 3 in map phase-II

Candi_pattern	Support	Candi_pattern	Support
ACT	1	CT	1
ATCG	3	CGATTG	1
ATTG	1	CGT	2
ATCGTGA	2	CGTCACT	1
ATCGTGAGA	1	CGTGAGA	1
AGA	1	CATCG	2
ATCGTGACT	1	CATCGATTG	1
ATCGATTG	1	CATCGTGAGA	1
TTG	1	GACT	1
TGA	2	GATTG	1
TGACT	1	GAGA	1
TGAGA	1	GTGA	2
TCGTGA	2	GTGACT	1
TCGTGACT	1	GTGAGA	1
TCGTGAGA	1	GA	3
TCGATTG	1	GTG	2

4.3.2 Reduce Phase

i. In Reduce phase-1, the master node assigns reduce task to idle worker nodes. Each worker node takes input as <candi_pattern, support> pairs and shares the support of contiguous candidate suffix sequences and combines the output after that emits the results as <freq_pattern, support> pairs and writes the output pairs in the local disks. Table 7 has shown the contiguous frequent patterns with corresponding supports as <freq_pattern, support> pairs; usually, it takes relatively less space since the number of contiguous frequent patterns set is small
ii. In reduce phase 2, worker 6 checks for the maximality criteria among the <freq_pattern, support > pairs and

Table 6: Intermediate values – From worker 4 in map phase-II

Candi_pattern	Support	Candi_pattern	Support
ACT	1	CT	1
ATTG	2	CG	3
ATTACT	1	CGATTG	1
AGTCGATTG	1	CGTGATT	2
ATT	2	CGTGATTG	1
TTG	2	CGTGATTACT	1
TTACT	1	GATT	2
TGATTG	1	GATTG	2
TGATT	2	GATTACT	1
TGATTACT	1	GT	2
TCG	2	GTGATTG	1
TCGTGATTG	1	GTCGATTG	1
TCGATTG	1	GCGTGATTACT	1

Table 7: Frequent pattern – Temporary output from worker 5 in reduce phase-I

Freq_pattern	Support	Freq_pattern	Support
AT	3	CG	6
ATCG	3	CGT	4
AG	2	CT	2
ACT	2	CATCG	2
ATCGTGA	2	CGTGATT	2
ATTG	3	CGATTG	2
TTG	3	GA	5
TGA	2	GTGA	2
TCG	6	GATT	2
TCGATTG	2	GATTG	3

therefore, writes the complete set of maximal contiguous frequent patterns in the output files. Table 8 has shown the maximal contiguous frequent patterns with corresponding supports as <max_pattern, support> pairs. Finally, we have six maximal contiguous frequent patterns, they are as follows: ACT, AG, ATCGTGA, TGATTG, CATCG, and CGTGATT.

5. Experimental Results

We used Hadoop version 0.20.0, running on a cluster with 6 machines (1 master, 5 workers). Master node has 3.7 GHz Intel Core 2 Duo processor with 4 GB of RAM and each worker machine has a processor with 2.60 GHz and 2 GB RAM. All programs were written in Java using MapReduce library functions and for this we configured the HDFS on Ubuntu-11.04. We applied our MapReduce framework on Human genome (Homo Sapiens GRCh37.64 DNA Chromosome Part 1) and Bacteria DNA sequence datasets downloaded from the NCBI website. The Human genome database contains 112 000 sequences; average length is 60; on the other hand, the Bacteria dataset consists of 20 000 sequences with average length of 1040 bp. Datasets were splited across 3 and 5 worker nodes for the first and second experiment respectively; and the load balancing was adopted from the literature [15].

We did not compare our results with any existing contiguous frequent pattern mining algorithm, since all of the previous works were implemented on main memory-based single processor machine. Figure 3 shows the running time of MCFP algorithm on Bacteria dataset and Figure 4 shows the running time of MCFP algorithm on Human genome dataset and from the graph, it is clear that our MapReduce is fast as well as scalable scalable in terms of increasing loads. We speed up the framework by adding more worker nodes. Figure 5 shows the running time after speeds up the framework by increasing worker nodes from 3 to 5.

6. Conclusion

In this paper, we have proposed an efficient approach for mining maximal contiguous frequent patterns using MapReduce on Hadoop. Our performance study shows that our MCFP algorithm can find the complete set of maximal contiguous frequent patterns very efficiently. The results also indicate the correctness and scalability in terms of increasing load. Since, mutations are essential to evolution; a mutation is a change in DNA. Hence, organism's DNA affects how it looks, how it behaves, and its physiology so, a change in an organism's DNA can cause changes in all aspects of its life. Therefore, in future our target is to extend this work by including gaps and execute it on real biological datasets.

Table 8: Maximal contiguous frequent patterns – From worker 6 in reduce phase-II

Max_pattern	Support
ACT	2
AG	2
ATCGTGA	2
TCGATTG	2
CATCG	2
CGTGATT	2

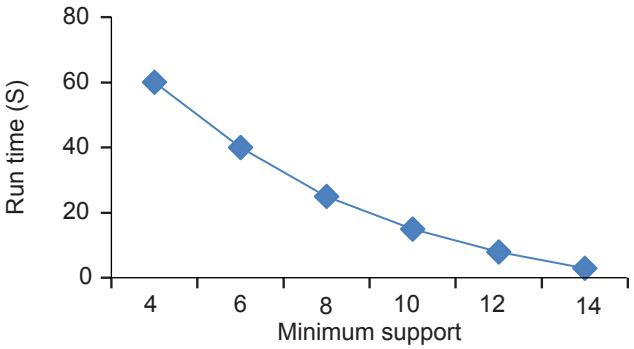

Figure 3: Runtime with the change of minimum support (bacteria dataset).

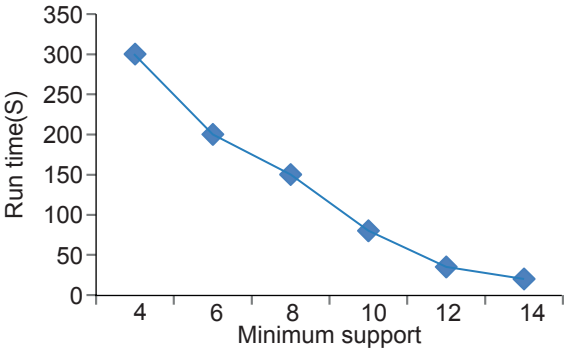

Figure 4: Runtime with change of minimum support (human genome).

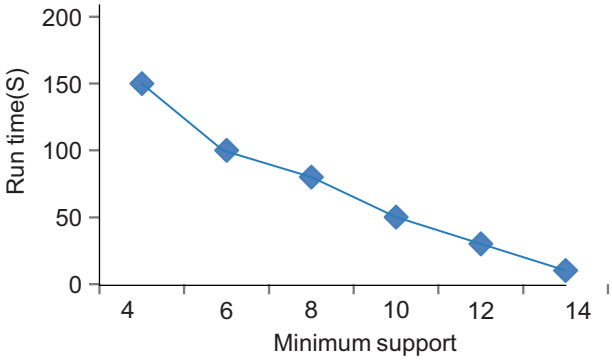

Figure 5: Speed up by increasing worker nodes (on human genome dataset with 5 worker nodes).

7. Acknowledgement

This work was supported by the National Research Foundation (NRF) grant (No. 2011-0018264) of Ministry of Education, Science and Technology (MEST) of Korea.

References

1. D. Hirschberg, "Algorithms for the longest common subsequence problem" (JACM, 1977).
2. S. Tata, R.A. Hankins, and J.M. Patel "Practical Suffix Tree Construction" 30th VLDB International Conference, Toronto, Canada, 2004.
3. H. Huo, and V. Stojkovic "A Suffix Tree Construction Algorithm for DNA Sequences", IEEE International. Conference, 2007.
4. R. Agrawal, and R. Srikant "Fast algorithms for mining association rules", 20th International Conference on VLDB, Santiago, 1994.
5. J. Pei, J. Han, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.C.Hsu "PrefixSpan: Mining sequential sequences efficiently by prefix-projected pattern growth", In proc. of the IEEE International Conference on Data Engineering, Germany, 2001.
6. J. Pan, P. Wang, W. Wang, B. Shi, and G. Yang, "Efficient Algorithms for Mining Maximal Frequent Concatenate Sequences in Biological Datasets" In proceedings of the 5th International Conference on Computer and Information Technology(CIT,2005), Shanghai, China, vol. 1. pp. 98-104, Sept. 2005.
7. T.H. Kang, J.S. Yoo, and H.Y. Kim, "Mining Frequent Contiguous Sequence in Biological Sequences", In Proc. of the 7th IEEE International Conference on Bioinformatics and Bioeng, 2007.
8. R. Chvatal, and D. Sankoff, "Longest Common Subsequences of two random Sequences" Journal of Applied Probability, Vol. 12, No. 2, pp. 306-315, Jun. 1975.
9. Available from: http://www.google.com/ [Last cited on 2011 Oct 15].
10. J. Dean, and S. Ghemawat, "MapReduce: Simplified data processing on large clusters", ACM, Jan. 2008
11. B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo, "PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce", Journal of the Proceedings of VLDB (PVLDB), Vol. 2, No. 2, pp. 1426-1437, Sep. 2009.
12. T. Elsayed, J. Lin, and D.W. Oard, "Pairwise Document Similarity in Large Collections with MapReduce," In Proceedings of the 32nd International ACM Conference on Research and Development Ret, 2009.
13. Available from: http://hadoop.apache.org/hdfs/ [Last cited on 2011 Oct 15].
14. J. Ekanayake, S. Pallickara, and G. Fox, "MapReduce for data intensive scientific analyses", In proceedings of the 4th IEEE International Conference on e-Science, pp.277-284, Indianapolis, USA, 7-12 Dec, 2008.
15. J. Xie, A. Manzanares, and X. Qin, "Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters" IEEE Intl. Symposium on PDPW and PhD Forum, 2010.

AUTHORS

Md. Rezaul Karim received the BS degree from the Dept. of Computer Science and Engineering, University of Dhaka, Bangladesh, in 2009. Currently, he is an MS degree candidate at the Dept. of Computer Engineering, Kyung Hee University, Korea. His research interest includes data mining, ubiquitous data management, and bioinformatics.

E-mail: asif_karim@khu.ac.kr

Md. Azam Hossain received the BS degree from the Dept. of Computer Science and Engineering, University of Dhaka, Bangladesh, in 2008. Currently, he is an MS degree candidate at the Dept. of Computer Engineering, Kyung Hee University, Korea. His research interest includes knowledge discovery, data mining, and database systems.

E-mail: azam@khu.ac.kr

Md. Mamunur Rashid received the BS degree from the Dept. of Electronics and Communication Engineering, Khulna University of Engineering and Technology, Bangladesh, in 2007. Currently, he is an MS degree candidate at the Dept. of Computer Engineering, Kyung Hee University, Korea. His research interest includes data mining, computation biology, molecular biology, and Bioinformatics.

E-mail: mamun@khu.ac.kr

Byeong-Soo Jeong received the BS degree in Computer Engineering from Seoul National University, Korea, in 1983. He received MS degree in Computer Science from the Korea Advanced Institute of Science and Technology in 1985, and the PhD in Computer Science from the Georgia Institute of Technology, Atlanta, in 1995. From 1996, he is a professor at the Department of Computer Engineering, Kyung Hee University, Korea. From 1985 to 1989, he was on the research staff at Data Communications Corporation, Korea. From 2003 to 2004, he was a visiting scholar at the Georgia Institute of Technology, Atlanta. His research interests include database systems, data mining, and mobile computing.

E-mail: jeong@khu.ac.kr

Ho-Jin Choi received the BS degree in Computer Engineering from Seoul National University, Korea, in 1985. He received MS in Computing Software and Systems Design from Newcastle University, UK and the PhD in Artificial Intelligence from Imperial College, London, in 1995. Between 1995 and 1996, he was a post-doctoral researcher at IC-PARC, Imperial College, London. From 1997 to 2002, he worked as an assistant professor of Computer Engineering at Korea Aerospace University. Currently, he is an associate professor at the Dept. of Computer Science at KAIST, Korea. His research interests include artificial intelligence, data mining, software engineering, and biomedical informatics.

E-mail: hojinc@kaist.ac.

DOI: 10.4103/0256-4602.95388; Paper No. TR 346_11; Copyright © 2012 by the IETE