Performance Analysis of Big Data Parallel Processing using Message Passing Interface

Mona Shukla [1], Paritosh Goldar [2],
[1,2]*Department of Computer Science & Engineering,*
RKDF University, Bhopal, India

***Abstract*—This paper presents an insight into the usage of Message Passing Interface for clustering of big data system. We processed parallel mining of big data frequent item set in transactional database of big data through MPI (Message Passing Interface) and MPI has been used here in distributed environment. Basically two algorithms used for mining in Distributed computing such as FP Growth and Apriori. FP Growth Algorithm has been parallelised through the use of MPI. FP Growth Algorithm is used for extracting significant data out of bulk of data. i.e. Data Mining. We have applied FP growth algorithm sequential as well as parallel and see the significant changes in time taken to mine the transaction. In parallel way, we used MPI to perform message communication. A significant feature is the time elapsed for processing. It draws out a comparison between time spent in parallel computing and single processor computing. On other hand, parallel algorithm has also been checked on performance basis.**

***Keywords*— Big Data, Data Mining, Transactional database, FP-Growth, MPI, Parallelization.**

I. Introduction

In this electronic age, increasing number of organizations are facing the problem of explosion of data and the size of the databases used in today's enterprises has been growing at exponential rates. Data is generated through many sources like business processes, transactions, social networking sites, web servers, etc. and remains in structured as well as unstructured form [1]. Today's business applications are having enterprise features like large scale, data-intensive, web-oriented and accessed from diverse devices including mobile devices. Processing or analyzing the huge amount of data or extracting meaningful information is a challenging task [1][2].

A. Big Data vs Traditional Data

Big data as "datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze". This definition is subjective and does not define big data in terms of any particular metric. However, it incorporates an evolutionary aspect in the definition (over time or across sectors) of what a dataset must be to be considered as big data [3]. Big data is where the data volume, acquisition velocity, or data representation limits the ability to perform effective analysis using traditional relational approaches or requires the use of significant horizontal scaling for efficient processing [4]. In particular, big data can be further categorized into big data science and big data frameworks. Big data science is "the study of techniques covering the acquisition, conditioning, and evaluation of big data", whereas big data frameworks are "software libraries along with their associated algorithms that enable distributed processing and analysis of big data problems across clusters of computer units". An instantiation of one or more big data frameworks is known as big data infrastructure [5].

TABLE I
Comparison between Big Data and Traditional Data

Comparision	Traditional Data	Big Data
Volume	GB	TB & PB
Generated Rate	per hour, day	more raid
Structure	Structured	Semi-structured or Un-structured
Data Source	Centralized	Fully Distributed
Data Integration	Easy	Difficult
Data Store	RDBMS	HDFS, NoSQL
Access	Interactive	Batch or Real-Time

Concurrently, there has been much discussion in various industries and academia about what big data actually means. However, reaching a consensus about the definition of big data is difficult, if not impossible. A logical choice might be to embrace all the alternative definitions, each of which focuses on a specific aspect of big data. The aforementioned definitions for big data provide a set of tools to compare the emerging big data with traditional data analytics. This comparison is summarized in Table I, under this framework, first, the sheer volume of datasets is a critical factor for discriminating between big data and traditional data [6].

II. Big Data Technology and Tools

In this section, technology and tools that have big impact on big data service are pointed out. First the explanation of hardware technologies followed by software technologies. Later in this section, brief discussion is presented on some tools that are made for different purposes in big data service.

A. Hardware Technology

Conventional storage technology DRAM to store persistent data faces problem for long-term use because disks have moving parts that are vulnerable to malfunction in long run. DRAM chips need constant power supply irrespective of its usage. So, it is not an energy-efficient technology. Non-volatile memory technology shows a promising solution in future memory designs [7][8]. There are thinkings on use

of NVM even at instruction level so that operating system can work fast. It is a wish to see NVM technology brings revolution to both data store and retrieval. Other than memory, technology looks for improving processing power to address the need for fast data processing. Significant solution towards that includes Data-Center-on-Chip (DOC) [9]. It proposes four usage models that can be used to consolidate applications that are homogeneous and cooperating and manage synchrony on shared resources and at the same time speed up computation providing cache hierarchies. Tang et al. [10] proposes a hardware configuration that speeds execution of Java virtual machine (JVM) by speeding up algorithms like garbage collection [10]. Same idea can be adopted for big data processing applying hardware technology to speed up data processing at bottlenecks usually found at data being shared by many.

Visualization technology though came with mainframe technology and gone low for availability of inexpensive desk top computing has come to forefront again for processing big data service on cloud environment [11]. Technologies are coming up for both CPU, memory and I/O visualization. For big data analytics, even code visualization (like JVM) is being intended. This technology helps in run-time visualization following dynamically typed scripting languages or the use of just-in-time (JIT) techniques [12].

B. Software Technology

This section take up developments in software technology taking place for big data services. First, it is pointed out the requirements in software technology in development of big data systems [13]. The requirements include storage management and data processing techniques particularly towards business intelligence applications.

Big data storage not only faces challenge in hardware technology but also the challenge with its store management [14]. As discussed through CAP theorem [15], indicates, assurance of high availability as well as consistency of data in distributed systems is always an ideal situation. Finally, one has to relax constraints in maintaining consistency. The nature of applications makes decisive impact assurance of consistency. Some applications need eventual consistency. For example, Amazon uses Dynamo [16] for its shopping cart. Facebook uses Cassandra [17] for storing its varieties of postings. Issues such as file systems, data structures and indexing are being actively studied for making big data systems meet growing user needs [18].

Data processing requirements can be better understood by following the way data are being generated in big data environment. Other than being bulk and heterogeneous, big data characterizes with its offline and online processing [19]. Streaming (online) data processing and that at back-end need different approaches as latency for both are diametrically different. Again taking application characteristics into consideration, we find task parallelism is required by scientific applications and data-parallelism by web applications. However, all data-centric applications need to be fault-resilient, scalable as well as elastic in resource utilization. MapReduce [20] technique is well known for data parallel model used for batch processing. Google's MapReduce [20] is the first successful use of big data processing. It provides scalable and fault-tolerant file system in development of distributed applications. The paradigm has two phases, viz. Map and Reduce. On input, programmer-defined Map function is applied on each pair of (Key, Value): list to produce list of (Key, Value, list) list as intermediate list. And in Reduce phase, another programmer-defined function is applied to each element of the intermediate list.

The paradigm supports run-time fault-tolerance by re-executing failed data processing [20]. Open-source version Hadoop [21] supporting MapReduce [20] paradigm is expected to be used for half of the world data in 2015 [55]. There are different versions of MapReduce technique to cater to different types of applications, e.g. for online aggregation and continuous queries, a technique is proposed accordingly. There are many extension to MapReduce, proposed in to support asynchronous algorithms. Some other extensions of MapReduce [20] to support different types of applications and assuring optimized performance by introducing relaxed synchronization semantics are also proposed.

III. Literature Review

A. Parallel Mining

Typically, the process of mining for Frequent Patterns is applied in large databases. For this reason, parallel versions of several algorithms have been adapted, as the sequential versions tend to be very time consuming [22]. Taking advantage of the computation power parallel machines provide, reduces significantly the time required to scan the dataset and generated Frequent Patterns. Most of the existing parallel algorithms are based on sequential algorithms as plenty of state-of-art algorithms exist out there. Sequential versions of algorithms used for the process of frequent item set mining appear to work sufficiently well for uni-processor architectures. As the bulk of the process emerges from the pattern generation stage, the parallel versions attempt to apply the computational power of parallel systems for distributing workload among multiple processors [23][22].

This section explains parallel versions of three Apriori-based algorithms namely Count Distribution, Data Distribution, Candidate Distribution, a parallel version of the FP-growth algorithm. For presentation purposes, assume P_i to be a processor with id.

B. The Count Distribution Algorithm

Count Distribution [24], is a parallel Apriori-based algorithm used to mine for frequent patterns. Each processor P_i computes its local candidate item set, along with their support count, by performing a single pass over its local data partition at that time. Information is maintained in a hash-table which is identical for each processor. This procedure is accomplished by running a sequential Apriori on each processor. All local counts are then accumulated and summed together to form a global support count using a global reduction function [25].

This is illustrated in Figure 1. Global reduction consists of two other operations. One of the operations is referred to as Reduce Scatter which is responsible for obtaining local support count communication from a processor Pi, and the other operation is named All Gather operation which is responsible for global support count communication. Count Distribution seems to

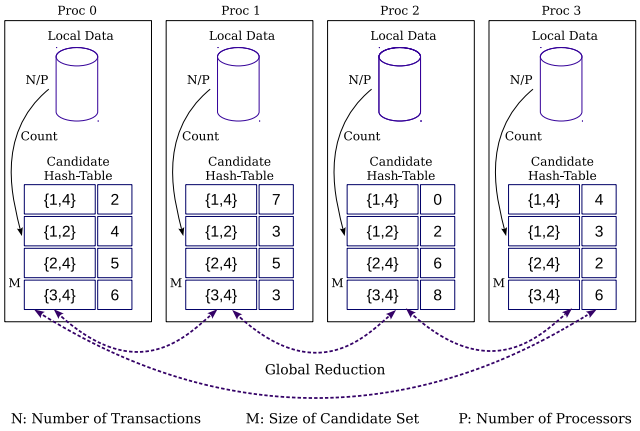

Fig. 1. The Count Distribution Algorithm

scale linearly to the number of records of the dataset as all computations to find support counts can be done locally at each processor having minor communication only at the end for accumulating the counts. However, in case the hash-table structure cannot fit into the main memory of each processor, it must be partitioned and support counts are computed by performing multiple scans of the dataset.

C. The Data Distribution Algorithm

The Count Distribution [24] algorithm is attractive in the sense that no data movement is performed. All the counts are computed locally to each processor thus every processor can operate asynchronously on its own data. However, this limits the ability of taking advantage of non-local memory parallel machines provide. The Data Distribution algorithm [26] solves this problem by allowing each processor to compute the support counts of its locally stored subset of the candidate item sets for all the transactions in the database. In order for this to become feasible, the All-to-All broadcast is used, where each processor must scan its own partition of the data as well as other partitioned data located at remote processors. This results in every processor having to broadcast their data to all other participating processors as well as receive data from them. Although this will solve the problem that Count Distribution carries with it, there are still negative effects as far as the burden placed in communication operations as there is a high communication overhead created due to data movement. Furthermore, such a communication scheme as this one causes the processors to become idle while waiting for data to be broadcasted resulting in wasting time that could have been manipulated for useful processing. The algorithm is illustrated in Figure 2.

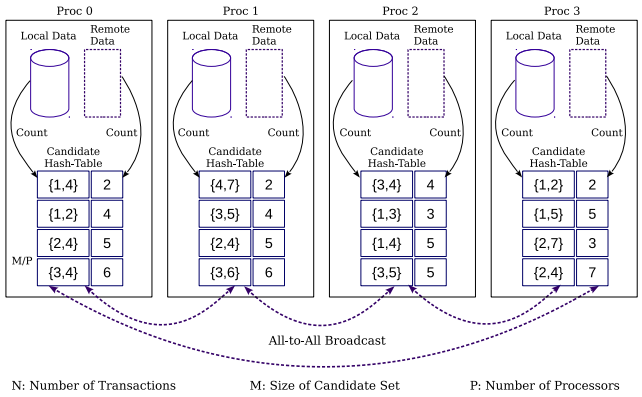

Fig. 2. The Data Distribution Algorithm

D. The Candidate Distribution Algorithm

Both Count [24] and Data distribution [26] algorithms carry the limitation that there is some synchronization involved. Although in Count Distribution, each processor can compute its own candidates asynchronously, some synchronization is required when global counts are about to be summed. In case the workload is not perfectly balanced some processors may have to remain idle until others are finished. Similarly, in Data Distribution, synchronization is needed when data is broadcasted around the processors. Furthermore, since any database transaction could support any candidate item set, each transaction must be compared against the entire candidate set. For this reason, Count Distribution needs to duplicate the dataset in every processor and Data Distribution needs to broadcast all of the transactions.

Candidate Distribution [27] combines the ideas used in both previous algorithms in order to overcome the problems associated with idle time, communication, and synchronization issues. This is achieved by duplicating the data on every processor's local memory as well as partitioning the candidate set across processors. In this way every processor can proceed independently, using its part of candidates on its local data. There is no need to exchange data or counts using this algorithm. The only communication required is when pruning a local candidate set during the phase of pruning in candidate generation. However, there is no need for synchronization at this stage thus no processor has to remain idle until pruning updates from other processors arrive.

E. Parallel Mining of Association Rules from Text Databases on a Cluster of Workstations

This section describes a parallel implementation of the serial Apriori algorithm described. As already mentioned the major bottleneck of Apriori-based algorithms arises from the fact that a large number of candidates may need to be generated [28]. In addition, multiple scanning of the database has a negative impact upon execution time. The Apriori algorithm adapts a different perspective, from Apriori-based ideas, by mining for frequent patterns avoiding the costly candidate generation phase. Also the use of a tree structure to store the transactions

circumvents the need of multiple scans. Furthermore, taking advantage of the aggregate memory and processing power of parallel machines, Apriori appears to achieve higher speedups compared to its sequential version as well as scalable performance, especially for very large datasets [29].

The serial version of the algorithm has been described in detail in next section; therefore, attention will be given only to parallel issues concerning this idea. The main idea is to make one main tree on master node and slave do processing with database rather than have multiple Apriori-Trees, one for each processor.

IV. Conclusion and Future Scope

In this review, the real time analytical and comprehensive analysis of big data parallelization techniques and methods are presented. Basic tools to process and execute of big data are summarized and its unique characterizations are investigated based on the analytical methods and experiments. The advanced and future approaches of big data are reviewed as well. The big data modules, properties and applications in business, research and industries presents broad future scope of this area. There are several further future analysis of big data parallelization when it is integrated with other core implementation of programming and algorithms.

V. Solution Approach

Basically there are estimated two approaches for the mining of data:

A. FP-Growth Algorithm

Let $i = fa_1; a_2; :::; a_n$ be a set of items, and a transaction database $DB = T_1; T_2; :::; T_n, whereT_i(i_2[1 :: n])$ is a transaction which contains a set of items in I. The support1 (or occurrence frequency) of a pattern A, which is a set of items, is the number of transactions containing in DB. A, is a frequent pattern if A's support is no less than a predefined minimum Support threshold. Given a transaction database DB and a minimum support threshold, the problem of finding the complete set of frequent patterns is called the frequent pattern mining Problem. The main steps of FP-Growth method are as follows:

1) Construct conditional pattern base for each node in the FP-Tree.
2) Construct conditional FP-Tree from each conditional pattern-base.
3) Recursively mine conditional FP-Trees and grow frequent patterns obtained so far.
4) If the conditional FP-Tree contains a single path, simply enumerate all the patterns.

Features of FP-growth Algorithm Feature extraction adopts a FP-Tree structure and FP-growth Mining method based on FP-Tree without candidate generation, which optimized from FP-growth algorithm. FP-growth is a basic algorithm of generating frequent patterns. FP-growth employs an iterative approach known as a level-wise search, where $k - item$ sets are used to explore $(k + 1)$-item sets. FP-growth is an influential algorithm for mining frequent item sets for Boolean association rules. In some application cases the FP-growth behave not as good as expect (i.e., need to repeatedly scan the item sets, inefficient, consuming abundant resource of CPU). FP-growth is optimized algorithm from FP-growth.

B. Result Analysis

Consideration of database (size=5577) Here, a database of size 5577 are taken transaction and perform mining of item serial as well as parallel. The corresponding result is shown in table. It is analyzed through graph. **Table between execution of time and threshold in serial and parallel algorithm** For big data item-sets, the parallel execution (using MPI) takes less time as compared to the serial execution (without MPI). For threshold value of 0.10, serial execution takes 0.104 seconds whereas parallel execution takes 0.048 seconds which is the advantage of parallel execution.

TABLE II
Execution Time in Serial and Parallel Algorithm.

Big Data Item-Set	Serial Execution (Seconds)	Parallel Execution (Seconds)
Threshold– 0.10	0.104	0.048
Threshold– 0.15	0.060	0.040
Threshold– 0.20	0.044	0.038

1) Graph Corresponding to Table: The figure 3 represents a graph of execution time for serial and parallel execution. For lower threshold values of big data item-sets (for example 0.10), the parallel execution time is very less as compared to serial execution. The blue bars in the graph represents the serial execution (FP-Growth algorithm without message passing interface) and orange bars in the graph represents the parallel execution (FP-Growth algorithm with using message passing interface). For higher threshold values (for example, greater than 0.20), the parallel execution time is some less as compared to serial execution.

The figure 4 represents the graph of execution time for

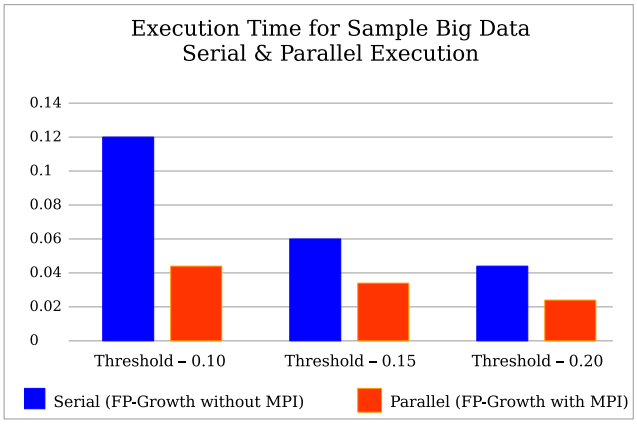

Fig. 3. Graph of Execution Time for Serial and Parallel Execution

MIHP (Multipass with Inverted Hashing and Pruning), PMIHP

(Parallel Multipass with Inverted Hashing and Pruning) and the proposed method APRIORI with MPI (message passing interface). The graph represents that the proposed method has less execution time as compared to previous methods MIHP (Multipass with Inverted Hashing and Pruning), PMIHP (Parallel Multipass with Inverted Hashing and Pruning). The MPI (message passing interface) uses task parallelization where as MIHP and PMIHP use data parallelization techniques. The task parallelization is more efficient than data parallelization in context of time complexity, so the proposed method presents better performance as compared to previous work.

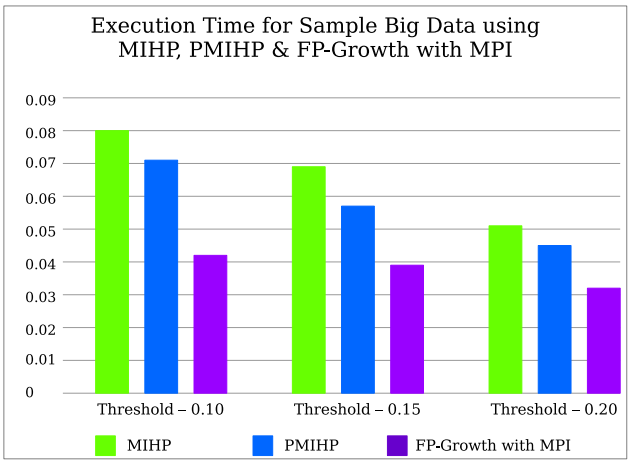

Fig. 4. Graph of Execution Time for MIHP, PMIHP and FP-Growth with MPI

C. Scalability Evaluation:

Serial run-time is indicated using T_s Parallel run-time using T_p.

1) Speed up:: Speed-up, S, is defined as the ratio of the serial run-time of the best sequential algorithm for solving a problem to the time taken by the parallel algorithm to solve the same problem on p processors:

$$S = \frac{T_s}{T_p}$$

2) Efficiency:: Efficiency E, is "a measure of the fraction of time for which a processor usefully employed"; it is defined as "the ratio of speed-up to the number of processors".

$$E = \frac{S}{P}$$

Here total number of processor core P = 3 (1 single core + 2 single dual core).

Result corresponding to Speed up and Efficiency For the given big data item-sets, the speedup is calculated for threshold value of 0.10, 0.15 and 0.20.

For lower threshold values, the proposed method presents high efficiency, however, for higher threshold values, it presents lower efficiency.

TABLE III
SPEED-UP AND EFFICIENCY

Big Data Item-Set	Threshold=0.10	Threshold=0.15	Threshold=0.20
Speed-Up	2.16	1.5	1.15
Efficiency	0.72	0.50	0.38

VI. CONCLUSION AND FUTURE WORK

Throughout the last decade, a lot of researchers have implemented and compared several algorithms that try to solve the frequent item set mining problem in web content mining as efficiently as possible. Moreover, we analyzed and experienced that different implementations of the same algorithms can still result in significantly different performance outcomes. As a consequence, several claims that were presented in some articles were later contradicted in other articles. In this paper, we performed web contents mining frequent item through MPI in transactional database for making difference between sequential and parallel algorithm, we noted the time of making APRIORI Tree for both algorithms on different threshold. After compare the result of different metric with idle parallel system metric, we conclude that our algorithm works good in parallel environment. MPI is used here for message communication between master and slave node as we made a paring here to create a virtual network. This method and technique helps MPI to communicate in parallel environment.

The future extension to this project may include some updates such as using optimization to make the database efficient. The database here comprised of numeric values. Instead, alphabetical letters could also be taken and then the scanning would be word by word. Also, most significantly, the processing time can be further compressed.

REFERENCES

[1] F. Z. Benjelloun, A. A. Lahcen, and S. Belfkih, "An overview of big data opportunities, applications and tools," in *Intelligent Systems and Computer Vision (ISCV), 2015*, March 2015, pp. 1–6.

[2] H. Eridaputra, B. Hendradjaya, and W. D. Sunindyo, "Modeling the requirements for big data application using goal oriented approach," in *Data and Software Engineering (ICODSE), 2014 International Conference on*, Nov 2014, pp. 1–6.

[3] H. Amir and R. Asim, "The emerging era of big data analytics," *Big Data Analytics*, vol. 1, no. 1, pp. 1–2, 2016.

[4] J.-L. Monino, "Data value, big data analytics, and decision-making," *Journal of the Knowledge Economy*, pp. 1–12, 2016.

[5] J. Ding, D. Zhang, and X. H. Hu, "A framework for ensuring the quality of a big data service," in *2016 IEEE International Conference on Services Computing (SCC)*, June 2016, pp. 82–89.

[6] L. Cao, "Data science and analytics: a new era," *International Journal of Data Science and Analytics*, vol. 1, no. 1, pp. 1–2, 2016.

[7] L. Wu, R. J. Barker, M. A. Kim, and K. A. Ross, "Hardware partitioning for big data analytics," *IEEE Micro*, vol. 34, no. 3, pp. 109–119, May 2014.

[8] P. Koutsourelakis, N. Zabaras, and M. Girolami, "The features, hardware, and architectures of data center networks: A survey,"

Journal of Parallel and Distributed Computing, vol. 96, pp. 45–74, October 2016.

[9] P. Bogdan, "Workload modeling and its implications on data-center-on-a-chip optimization: From mathematical models to control algorithms," in *2015 20th International Conference on Control Systems and Computer Science*, May 2015, pp. 1001–1001.

[10] J. Tang and C. Liu, "An energy and memory trade-off study on resource constrained embedded jvm," in *2014 43rd International Conference on Parallel Processing Workshops*, Sept 2014, pp. 448–452.

[11] S. P. Menon and N. P. Hegde, "A survey of tools and applications in big data," in *Intelligent Systems and Control (ISCO), 2015 IEEE 9th International Conference on*, Jan 2015, pp. 1–7.

[12] J. S. Saltz, "The need for new processes, methodologies and tools to support big data teams and improve big data project effectiveness," in *Big Data (Big Data), 2015 IEEE International Conference on*, Oct 2015, pp. 2066–2071.

[13] J. Rekha and R. Parvathi, "Survey on software project risks and big data analytics," *Procedia Computer Science*, vol. 50, pp. 295–300, 2015.

[14] G. Iuhasz and I. Dragan, "An overview of monitoring tools for big data and cloud applications," in *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Sept 2015, pp. 363–366.

[15] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, Feb 2012.

[16] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 205–220. [Online]. Available: http://doi.acm.org/10.1145/1294261.1294281

[17] A. Lakshman and P. Malik, "Cassandra: A structured storage system on a p2p network," in *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '09. New York, NY, USA: ACM, 2009, pp. 47–47. [Online]. Available: http://doi.acm.org/10.1145/1583991.1584009

[18] C. K. Emani, N. Cullot, and C. Nicolle, "Understandable big data: A survey," *Computer Science Review*, vol. 17, pp. 70–81, August 2015.

[19] M. Chen, S. Mao, and Y. Liu, "Big data: A survey," *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014.

[20] J. Ullman, "Mapreduce algorithms," in *Proceedings of the 2Nd IKDD Conference on Data Sciences*, ser. CODS-IKDD '15. New York, NY, USA: ACM, 2015, pp. 1:1–1:1. [Online]. Available: http://doi.acm.org/10.1145/2778865.2778866

[21] S. Bende and R. Shedge, "Dealing with small files problem in hadoop distributed file system," *Procedia Computer Science*, vol. 79, pp. 1001–1012, 2016.

[22] M. Nagao and H. Seki, "Towards parallel mining of closed patterns from multi-relational data," in *2015 IEEE 8th International Workshop on Computational Intelligence and Applications (IWCIA)*, Nov 2015, pp. 103–108.

[23] L. Xu and Z. Yun, "A novel parallel algorithm for frequent itemset mining of incremental dataset," in *Information Science and Control Engineering (ICISCE), 2015 2nd International Conference on*, April 2015, pp. 41–44.

[24] T. Wen, G. Wang, Q. Guo, and X. Ma, "An optimal association rule mining algorithm based on knowledge grid," in *Fuzzy Systems and Knowledge Discovery, 2008. FSKD '08. Fifth International Conference on*, vol. 2, Oct 2008, pp. 572–575.

[25] T. Marschall and S. Rahmann, "An algorithm to compute the character access count distribution for pattern matching algorithms," *Algorithms*, vol. 4, no. 4, p. 285, 2011.

[26] J. Zhou, W. Xie, J. Noble, K. Echo, and Y. Chen, "Suora: A scalable and uniform data distribution algorithm for heterogeneous storage systems," in *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*, Aug 2016, pp. 1–10.

[27] C. X. C. Xu, F. Y. F. Yu, Z. D. Z. Dai, G. Y. G. Yue, and R. L. R. Li, "Data distribution algorithm of high-speed intrusion detection system based on network processor," in *Semantics, Knowledge and Grid, 2006. SKG '06. Second International Conference on*, Nov 2006, pp. 27–27.

[28] L. Troiano, G. Scibelli, and C. Birtolo, "A fast algorithm for mining rare itemsets," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*, Nov 2009, pp. 1149–1155.

[29] S. Shankar, N. Babu, T. Purusothaman, and S. Jayanthi, "A fast algorithm for mining high utility itemsets," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, March 2009, pp. 1459–1464.