Parallelization of Frequent Itemset Mining Methods with FP-tree: An Experiment with PrePost$^+$ Algorithm

Olakara Jamsheela[1] and Raju Gopalakrishna[2]
[1]EMEA College of Arts and Science, Calicut University, India
[2]Computer Science and Engineering, CHRIST (Deemed to be University), India

Abstract: *Parallel processing has turn to be a common programming practice because of its efficiency and thus becomes an interesting field for researchers. With the introduction of multi- core processors as well as general purpose graphics processing units, parallel programming has become affordable. This leads to the parallelization of many of the complex data processing algorithms including algorithms in data mining. In this paper, a study on parallel PrePost+ is presented. PrePost+ is an efficient frequent itemset mining algorithm. The algorithm has been modified as a parallel algorithm and the obtained result is compared with the result of sequential PrePost+ algorithm.*

Keywords: *Data Mining algorithm, parallelization of PrePost+, parallel processing, multicore.*

Received April 6, 2020; accepted August 26, 2020
https://doi.org/10.34028/iajit/18/2/9

1. Introduction

Parallel and distributed processing are inevitable components in big data processing. Parallel versions of Association Rule Mining as well as Frequent Itemset Mining have been developed in the last decade. A parallel version of the Pre- Post+, an FP-tree based Frequent Pattern Mining algorithm is presented in this paper. When considering the question "what is big data?" size is the first thing that comes to mind. However, other features of big data have been emerged recently [6]. Parallel processing is an efficient technique to reduce the processing time of big data.

1.1. Parallel Processing

Parallel processing is the processing of program instructions by dividing them among multiple processors with the objective of running a program in less time [15]. By using parallel processing, different parts of a single program can be run simultaneously. The running time can be reduced. Each processor works independently and processors communicate each other using some specific functions or with shared memory. The commonly used hardware platforms for parallel processing includes multicore, Graphics Processing Unit (GPU) and clusters.

1.1.1. Multicore

Recently, integrated circuits are designed to put several CPUs on one chip, termed as multicore chip. Dual-core chips and quad-core processors are now available at affordable cost. As the invention of the integrated circuits revolutionized the computer industry by making computers affordable to the man, multicore chips undoubtedly revolutionized the world of parallel programming [17] Parallel programs can run in a single machine with a multi-core processor. The bottleneck is the size of the main memory and the cost associated with memory management. When the memory requirement is very high, cluster computing is a cheaper and efficient alternative.

1.1.2. Clusters

A group of individual PCs with uniprocessor (or multiprocessor) are connected together by using a network and using parallel-processing software, can form very powerful parallel processing system. The most common type of cluster is the Beowulf cluster, which is implemented on multiple identical commercial off-the-shelf computers connected with a Transmission Control Protocol and Internet Protocol (TCP/IP) Ethernet local area network [8]. The majority of the TOP500 supercomputers are clusters [8].

1.1.3. GPU

GPU is a special purpose processor designed for calculations required for Computer Graphics. GPU contains thousands of smaller and efficient cores optimized for handling multiple tasks simultaneously. A CPU is a general purpose processor consists of a few cores optimized for sequential serial processing. CPU can do any computation including Graphics processing but not in an optimal fashion and cannot produce the result as fast as a GPU. Both can work together while GPU is doing calculations for graphics, CPU can do other non-graphics calculations simultaneously. Parallel programming can be implemented with single GPU but multiple GPUs can be used for higher levels of parallelism. GPU and CPU can be used together in a single system for parallel programming or both can be connected together as clusters in separate systems. The combination can deliver the best performance. an American technology company (NVIDIA) has developed the Compute Unified Device Architecture (CUDA) language as a vehicle for programming on their GPUs. CUDA enables to harness the tremendous computational power and memory bandwidth of the GPU in a familiar programming environment [14].

1.1.4. Parallel Processing with MATLAB

MATLAB [21] is one of the most widely used interactive mathematical computing environments in technical computing. It provides high performance computational routines and an easy-to-use, C-like scripting language [4]. Parallel Computing Toolbox (PCT) of Matlab enables to create parallel Matlab programs using multicore processors, GPUs, and computer clusters. High level constructs- parallel for-loops, special array types, and parallelized numerical algorithms support to parallelize MATLAB applications without CUDA or MPI programming [16]. If multiple clusters are used, then use PCT functions and MATLAB Distributed Computing Server (MDCS) software. The method ‘matlabpool’ is used to indicate the starting of the parallelization of the program and also used to set the number of workers or clusters. MATLAB*P [10] provides a user-friendly environment to implement parallelism in MATLAB with the use of object oriented programming features. Among the most notable MATLAB based utilities for parallel programming are pMATLAB [22] and MatlabMPI [12] from Massachusetts Institute of Technology (MIT) Lincoln Laboratory.

1.1.5. Parallel Programming with Java

Java Platform, Java SE 5 and Java SE6 introduced a set of packages providing powerful concurrency building blocks called java.util.concurrent. Java SE 7 further enhanced them. Earlier versions of Java support multithreaded programming. Many new features such as Executors, thread safe queues, rich synchronization patterns, a wide range of locks etc. have been introduced in this package to improve the performance of parallel programming. The fork/join framework is added in Java SE 7 to support the parallelization of divide- and-conquer algorithms [23].

Open source software are also available for parallel processing. MPJ Express is an open source Java message passing library that allows application developers to write and execute parallel applications for multicore processors and compute clusters/ clouds [20]. Apache Hadoop project develops open-source software for reliable, scalable, distributed computing [19]. The Apache Hadoop software library allows the distributed processing of large data sets across clusters of computers using simple programming models. The library is designed to detect and handle failures at the application layer, on top of a cluster of computers [9]

1.1.6. The Exact Time to Use Parallel Processing

The complex and time consuming programs can be made easy and fast by using parallel programming. Many real time problems are so large to solve them on a single computer as a sequential program with limited computer memory. In these situations the parallel processing is the best solution. But not every task runs better in parallel. Parallel processing is much more complex than corresponding serial applications. Multiple instruction streams have to be separately assigned to each processor. The communication among the processors should also be well managed. Message passing among distributed memory may degrade the efficiency of the process. Some programs such as short running programs, programs with no independent executable units etc. perform better in serial implementation than parallel implementation. The overhead costs associated with communications, setting up the parallel environment, task creation and termination can comprise a significant portion of the total execution time for such programs.

2. Literature Review

Very few research papers have been published suggesting the use of parallel processing in Frequent Itemset Mining. Researchers finds that, when applying parallel processing on Apriori algorithms, it suffer I/O and synchronization overhead. But they suggested that the problem can be solved by using the FP-growth like methods. The authors of International Advance Computing (IAC) algorithm, use CR-tree and FP-tree to generate hidden rules instead of using the Apriori [2]. FP-growth method is faster than Apriori [11], but FP-growth algorithm also have some problems with parallel processing such as the construction of in-memory FP-tree. Athavale *et al*. [3] suggest a tool to convert the sequential C programs to parallel code. Another method is introduced by Mohemmad and Refaat [18] by suggesting to divide the database and distributes it over the system nodes. Xia *et al*. [24] suggested a parallel frequent itemset mining algorithm called IPFP by using Hadoop MapReduce to handle massive small files datasets effectively. A parallel association rule mining method based on cloud computing is implemented by Yong *et al*. [27]. A new method called Fidoop is proposed for parallel processing of frequent itemset mining by using frequent ultrametric trees (FIUT) instead of FP-tree. The MapReduce programming model is used to develop the parallel frequent itemsets mining algorithm [25]. Parallel Association Rules Extractor from SNPs (PARES) is introduced as a novel parallel algorithm for the efficient extraction of association rules from omics datasets [1]. Another paper proposed a search strategy for Frequent Itemset Mining (FIM), based on equivalence classes partitioning which allows dividing the search space into disjoint sets and enables parallel processing [13]. A Parallel Particle swarm optimization for Quantitative Association Rule mining (PPQAR) is proposed with two methods, particle-oriented and data-oriented parallelization [26].

It can be noticed that most of the papers used the parallel technology to apply in a specific field and proved as efficient. The data sets are also specific and the algorithms applied in specific fields. The authors did not mention the common steps to follow while converting into parallel algorithm. These are the major issues detected in the related works. In this paper we have mentioned the detailed procedure and steps to convert a sequential algorithm into a parallel one which can be applied on any algorithm and experimentally applied the method on a popular FP-tree based association rule mining algorithm, PrePost$^+$. The experimental results show that the proposed method is efficient than sequential algorithm.

3. FP-Tree Based Parallel Algorithm For Frequent Itemset Mining

When converting a sequential program to parallel program, the programmer has to think how to break the programming instructions into pieces, and has to figure out how the pieces relate to each other. The efficiency of the method greatly depends on the structuring of the program in each processor. In this paper the recent frequent itemset mining algorithm, PrePost+ [5], is selected to introduce the efficiency of the parallelization. The implementation details are described below.

3.1. Parallel PrePost+

The FP-growth algorithm have mainly three phases.

1. The pre-processing step: the processes applied on the transactions before inserting into the FP-tree.
2. Tree construction. The processed transactions have to be inserted to the FP-tree data structure.
3. The mining phase. The frequent itemsets are mined from the FP-tree.

The above three steps are processed sequentially. The tree construction starts only after completing the first phase and the mining process starts only after the completion of tree construction. It is not desirable to assign each phase to each processor as each processor have to wait for the output of the other processor. To get the maximum efficiency of parallel processing the set of programming instructions within these three phases can be split and allotted to different processors. In the proposed approach, only the first phase is converted to a parallel code.

The algorithm PrePost+suggested by Deng *at el*., [5] which is an efficient Frequent Itemset mining algorithm, is chosen for parallelizing. The algorithm starts by scanning the database to find the frequent 1-itemsets. Next step is the transaction processing. To process each transaction, scan the database again, remove the infrequent items from the transaction and sort the frequent items. The tree construction is carried out by taking each processed transaction.
The steps in the design of parallel PrePost+ are:

1. Split the database based on the number of cores (processors). If *n* cores are used, split database into *n*.
2. Each core is assigned function segment to find frequent 1-itemsets.
3. Load code segment for transaction processing into the *n* cores. Processes other than (2) and (3) are assigned to the main core Cm
4. Each core count the frequent 1-itemset by scanning the chunk of database allocated to it.
5. Each core forwards the frequent 1-itemsets with support count to a designated core, Cd. Cd merge the output of different cores and send the merged itemset with support count to all other cores.
6. After receiving the frequent 1-itemsets of the entire database, transaction processing is carried out in each core simultaneously.
7. Each core send the reduced transactions to the main core, Cm.
8. After completing all the transactions processing, the core Cm starts the tree construction.
9. The remaining procedures are done sequentially by Cm.

The block diagram in Figure 1 shows the complete procedure. We have carried out experiments with 3, 5 and 6 cores. Because of the overhead associated with splitting and merging, it is observed that the time required to complete the task is minimum when 3 cores are employed. Of the three, any one can be set as main core.

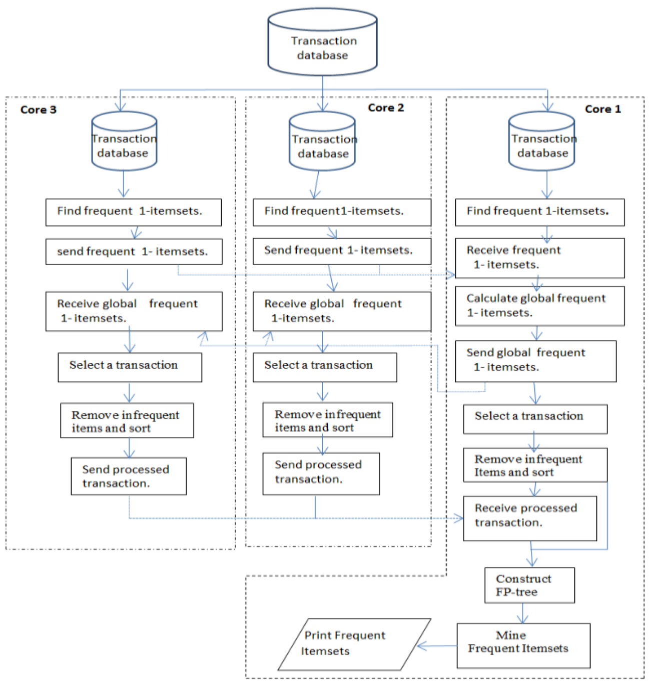
Transaction database
Core 3
Core 2
Core 1
Transaction database
Find frequent 1-itemsets.
send frequent 1- itemsets.
Send frequent 1- itemsets.
Receive frequent 1- itemsets.
Receive global frequent 1- itemsets.
Calculate global frequent 1- itemsets.
Send global frequent 1- itemsets.
Select a transaction
Remove infrequent items and sort
Send processed transaction.
Receive processed transaction.
Construct FP-tree
Mine Frequent Itemsets
Print Frequent Itemsets

Figure 1. The procedure of parallel PrePost+.

3.2. Problems with Parallelization of the Algorithm

Here, only the pre-processing phase is parallelized. The time consuming part of the suggested parallel process is the message passing. Each core uses its own local memory. To send each processed transaction a ‘send function’ should be invoked. The tree creation is done only in main core Cm. The Cm invoke the ‘receive method’ to receive all the processed transactions. To avoid this overhead a portion of the transaction can be assigned to Cm. The Cm simultaneously inserts the processed transactions into the Tree.

3.3. Parallelization of the Other Two Phases

The proposed method is implemented only on the transaction preprocessing phase. The tree creation phase and the mining phase can also be parallelized. Each core constructs its own local FP-tree and sends it to the main process. So the message passing can be reduced. But the local tree should be combined before starting the mining process and the global frequent itemset list should be prepared and broadcast. The mining phase can efficiently be converted to parallel code because mining of each item can be done independently with each other. A set of frequent items can be allocated to each core to mine its frequent itemsets. But the full FP-tree should be loaded to the local memory of each core.

4. Experimental Evaluation

In the experimental evaluation 5 datasets have been used. These datasets are publicly available datasets downloaded from FIMI repository (http://fimi.ua.ac.be) [7]. Retail is sparse data sets. The Mushroom, Connect, Pumsb, accidents etc. are dense data sets. The details of the datasets are shown in Table 1. The datasets accidents double is the double sized dataset of original accidents dataset. All other datasets have been used as its original form without losing any data. The data sets are real datasets.

Table I. Datasets.

	Data sets	Transactions	Items
1	Accidents	340183	468
2	Accidentsdouble	680366	468
3	Mushroom	8124	119
4	Pumsb	49046	2113
5	Retail	88162	16470

4.1. PrePost+ Algorithm

PrePost+ algorithm is a modification of PrePost. In PrePost+ the N-list data structure and N-list intersection of PrePost method to find frequent k-itemsets have been applied. The set enumeration tree is used in PrePost+ to speed up the mining process. The authors tried to bring together the advantages of PrePost and the advantages of FIN in PrePost+ to introduce a more efficient algorithm to mine frequent itemsets. The runtime of PrePost+ is compared with other 3 efficient algorithms and proved that PrePost+ is the fastest one among all algorithms for different minimum supports [5].

4.2. PRL-PrePost+ Algorithm

The PrePost+ algorithm is converted as Parallel PrePost+ by using the MPJ Express software library. The performance of the proposed method is evaluated by comparing with the sequential PrePost+ algorithm. The parallel PrePost+ is denoted as PRL-PrePost+ in the experimental results. The results are shown in Figures 2 to 6. From the experimental result it is observed that, when minimum support reduces the runtime of the parallel PrePost+ is less than the runtime of PrePost+ algorithm. The runtime of both of the algorithms with each dataset are displayed in Tables 2 to 6. Runtime is calculated in seconds. All the minimum support value of used during the execution of the algorithm with the dataset accidents double could not be included in the table. The algorithm is most effective with the dataset pumsb, which is a dense datasets. Other datasets are sparse datasets.

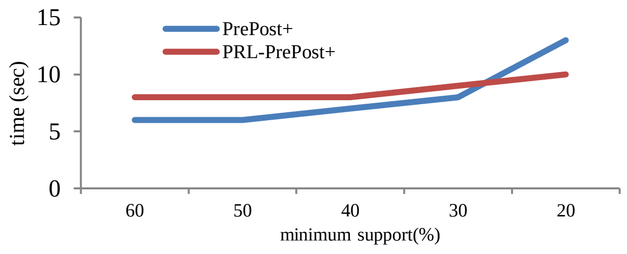

Figure 2. The runtime evaluation with dataset accidents.

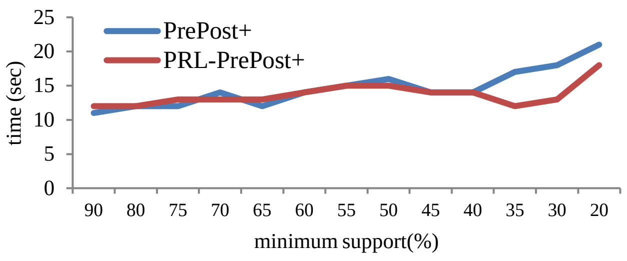

Figure 3. The runtime evaluation with dataset accidents double.

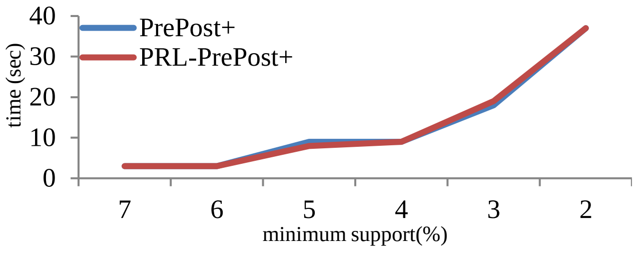

Figure 4. The runtime evaluation with dataset mushroom.

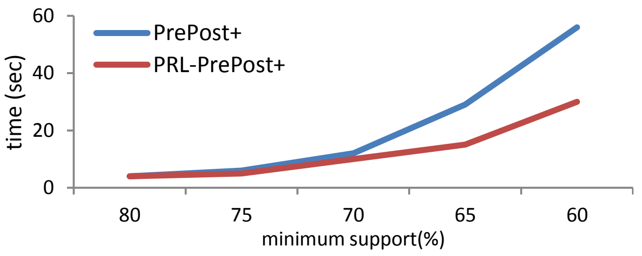

Figure 5. The runtime evaluation of Parallel PrePost+ with dataset pumsb.

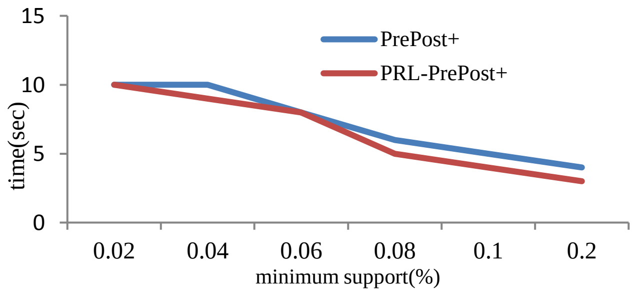

Figure 6. The runtime evaluation with dataset retail.

Table 2. Runtime with accidents dataset.

Min_support%	60	50	40	30	20
PrePost+	6	6	7	8	13
PRL-PrePost+	8	8	8	9	10

Table 3. Runtime with accidents double.

MinSupport%	55	50	45	40	35	30	20
PrePost+	15	16	14	14	17	18	21
PRL-PrePost+	15	15	14	14	12	13	18

Table 4. Runtime with mushroom dataset.

MinSupport%	7	6	5	4	3	2
PrePost+	3	3	9	9	18	37
PRL-PrePost+	3	3	8	9	19	37

Table 5. Runtime with pumsb dataset.

MinSupport%	80	75	70	65	60
PRL-PrePost+	4	5	10	15	30
PrePost+	4	6	12	29	56

Table 6. Runtime with retail dataset.

MinSupport%	80	75	70	65	60
PRL-PrePost+	4	5	10	15	30
PrePost+	4	6	12	29	56

The major limitations of the proposed method are the selection of the number of cores and the extra effort for the division of the dataset. Sequential algorithm is better if the minimum support value is very high.

5. Conclusions

With the recent developments in hardware and software, parallel processing has become essential component in computationally intensive tasks. Any sequential program code can be modified in to parallel programming code if it has independent program units. Here, parallel processing is explained in detail and the steps to design a parallel code has been introduced by using an example. The example is included to simplify and to clear the method in a practical approach. A parallel frame work for Frequent Itemset mining is proposed, where processing step of PrePost$^+$ is parallelized. The results obtained are encouraging.

References

[1] Agapito G., Guzzi P., and Cannataro M., "Parallel Extraction of Association Rules from Genomics Data," *Applied Mathematics and Computation*, vol. 350, pp. 434-446, 2019.

[2] Al-Fayoumi M., Alwidian J., and Abusaif M., "Intelligent Association Classification Technique for Phishing Website Detection," *The International Arab Journal of Information Technology*, vol. 17, no. 4, pp. 488-496, 2019.

[3] Athavale A., Randive P., and Kambale A., "Automatic Parallelization of Sequential Codes Using S2p Tool and Benchmarking of The Generated Parallel Codes," URL http://www. kpit. com/downloads/research-papers/automatic-parallelization-sequential-codes. pdf. Last Visited, 2020.

[4] Choy R. and Edelman A., "Parallel MATLAB: Doing it RIght," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 331-341, 2005.

[5] Deng Z. and Lv S.," PrePost+: An Efficient N-Lists-Based Algorithm for Mining Frequent Itemsets via Children-Parent Equivalence Pruning," *Expert Systems with Applications*, vol. 42, no. 13, pp.5424-5432, 2015.

[6] Gandomi A. and Haider M., "Beyond The Hype: Big Data Concepts, Methods, and Analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137-144, 2015.

[7] Goethals B., Fimi repository website. http://fimi.ua.ac.be/data/., Last Visited, 2020.

[8] Guo R., "Variance-Covariance Matrix Estimation with LSQR in A Parallel Programming Environ-Ment," Master's Thesis, 2008.

[9] Hadoop A., Welcome to Apache TM Hadoop®. URL http://hadoop.apache.org/, Last Visited, 2020.

[10] Husbands P., Isbell C., and Edelman A., "MITMatlab: A Tool for Interactive Supercomputing, *in Proceedings the 9th SIAM Confrence*, San Antonio, 1999.

[11] Jamsheela O. and Gopalakrishna R., "Frequent Itemset Mining Algorithms: A Literature Survey," *in Proceedings of IEEE International Advance Computing Conference* Banglore, pp. 1099-1104, 2015.

[12] Kepner J. and Ahalt S., "MatlabMPI," *Journal of Parallel and Distributed Computing*, vol. 64, no. 8, pp. 997-1005, 2004.

[13] Letras M., Bustio-Martínez L., Cumplido R., Hernández-León R., and Feregrino-Uribe C., "On the Design of Hardware Architectures for Parallel Frequent Itemsets Mining," *Expert Systems with Applications*, vol. 157, 2020.

[14] Luebke D., "CUDA: Scalable Parallel Programming for High-Performance Scientific Computing," *in Proceedings of 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, Paris, pp. 836-838, 2008.

[15] Lutke bohle I., Search Data Center, http://searchdatacenter.techtarget.com/definition/parallelprocessing/, Last Visited, 2020.

[16] Mathworks.Com: Parallel Computing Toolbox, MATLAB, MathWorks India, http://in.mathworks.com/products/parallel-computing/ Last Visited, 2020.

[17] Matloff N., *Programming on Parallel Machines*. University of California, 2011.

[18] Mohamed M. and Refaat H., "A Fast Parallel Association Rule Mining Algorithm Based on the Probability of Frequent Itemsets," *International Journal of Computer Science and Network Security*, vol. 11, no. 5, pp. 152-162, 2011.

[19] Patel A., Birl M., and Nair U., "Addressing Big Data Problem Using Hadoop and Map Reduce," *in Proceedings of Nirma University International Conference on Engineerings*, Ahmedabad, pp. 1-5, 2012.

[20] Shafi A., Carpenter B., and Baker M., "Nested Parallelism for Multi-Core HPC Systems Using Java," *Journal of Parallel and Distributed Computing*, vol. 69, no. 6, pp. 532-545, 2009.

[21] TECH, OF: MATLAB* P 2.0: Interactive Supercomputing, Massachusetts Institute of Technology, Dissertation, 2002.

[22] Travinin B. and Kepner J., "pMATLAB Parallel MATLAB Librar," *The International Journal of High Performance Computing Applications*, vol. vol. 21, no. 3, pp. 336-359, 2007.

[23] Tsagkli S., Ilias:Java Fork/Join for Parallel Programming Java Code Geeks-2016, https://www.javacodegeeks.com/2011/02/ java-forkjoin-parallel-programming.html, Last Visited, 2020.

[24] Xia D., Zhou Y., Rong Z., and Zhang Z., "IPFP: An Improved Parallel FP-Growth Algorithm for Frequent Itemsets Mining," *in Proceedings 59th ISI World Statistics Congress*, Hong Kong, pp. 4034-4039, 2013.

[25] Xun Y., Zhang J., and Qin X., "Fidoop: Parallel Mining of Frequent Itemsets Using Mapreduce," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 313-325, 2015.

[26] Yan D., Zhao X., Lin R., and Bai D., "PPQAR: Parallel PSO for Quantitative Association Rulemining," *Peer-to-Peer Networking and Applications*, vol. 12, no. 5, pp.1433-1444, 2019.

[27] Yong W., Zhe Z., and Fang W., "A Parallel Algorithm of Association Rules Based on Cloud Computing," *in Proceedings 8th International Conference on Communications and Networking in China*, Guilin, pp. 415-419, 2013.

Olakara Jamsheela received the M.Sc and Ph.D. in computer science and L.L.B(law) from the University of Kannur, Kerala,India in 2005,2016, 2002 respectively. She is currently working as Assistant Professor with the dept of Computer Science, EMEA College of Arts and Science, Kondotty affiliated to University of Calicut, Kerala, India. Her research interests include association rule mining and frequent itemset mining algorithms and also interested in the application of the data mining methods in different fields like health, medical field, users' behavior in social media etc.

Raju Gopalakrishna received the MCA and Ph.D. in computer science from the University of Kerala, Kerala,India in 1992, 2003 respectively. He is currently working as Professor with the dept of Computer Science and Engineering, Christ University, Bangalore, India. His research interests include Data mining, web mining and Image Processing.