International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 /doi:10.14445/22315381/IJETT-V69I1P203
Volume 69 Issue 1, 17-23, January 2021

Performance Comparison of Absolute High Utility Itemset Mining (AHUIM) Algorithm for Big Data

Sandeep Dalal[1], Vandna Dahiya[2]

[1,2]*DCSA, Maharshi Dayanand University, Rohtak, Haryana, India*

[1]sandeepdalal.80@gmail.com, [2]vandanadahiya2010@gmail.com

Abstract - *High utility itemset mining (HUI) targets the mining of high utility itemsets from a database. The utility here is defined as the amalgamation of the magnitude of the item and its importance. Although various studies have been done on HUI, they are mainly dedicated to centralized datasets and are not mountable for big data. A novel technique called the Absolute High Utility Itemset Mining (AHUIM) algorithm for parallel mining of HUIs has been recommended to tackle the issue of big data environment. The algorithm uses the Spark-in-memory computing architecture where the whole mining task is divided into smaller independent sub-tasks. Several pruning strategies have been used to implement the algorithm to efficiently mine the dataset, diminishing the need for traversing unpromising search space. The proposed algorithm inherits Spark's numerous properties such as fault tolerance, scalability, low communication cost, etc. In this research work, the functioning of AHUIM is being evaluated by comparing it with the most recent and fast algorithms for mining HUIs from big data. Extensive experiments show that the novel algorithm is better than other state-of-the-art algorithms for various factors such as time complexity, storage, scalability, etc.*

Keywords - *big data mining, distributed computing, MapReduce, Spark platform, utility mining*

I. INTRODUCTION

High Utility Itemset (HUI) mining is a method that discovers the set of items that occur together with a high utility value. It can be seen as a specialization of frequent itemset mining (FIM) as in FIM, all the items are considered as equal with the same utility or profit. Whereas FIM might ends up discovering the itemsets, which may be frequent but of little profit or use, HUIM discovers the itemsets with utility greater than a pre-specified minimum threshold. The utility is composed of two factors - internal utility (amount) and external utility (importance factor). This study area has been focused of research for many years due to various scientific domains and business purposes extending from gene regulated configurations to market basket analysis and customer behavior evaluation. For example, from the shopping scenario, in addition to suggesting the correlated items to the customers, HUIM plays a significant role in recommending various placements of items that can be put together in the stores to attract the customers. It helps in advocating various cross products to the customers based on their web-click streams. Various recommender systems can be developed based on the user's search history, such as music playlist generator, academic paper recommendation system, etc.

The rapid evolution of data generated from diverse sources such as enterprises, sensors, social networking, and the medical field has escorted us to the era of big data. Due to this huge data composed of diverse dimensions, the HUIM techniques have to deal with enormous search space, and mining is not efficient or excessively expensive on a particular machine. It can be understood with the instance of Walmart. In 2012, Walmart generated 2.5 petabytes of data in connection to one million users in one hour. From such a massive dataset, it becomes very difficult to study and discover the profitable patterns for the business. These patterns are important for maximizing the revenue and to find out the inventory costs.

Another example is from an online news portal where around 130 million visitors visited the portal page in 2015. To improve the business, there is a need to improve the design of the portal and recommend news based on their reading behavior. Traditional methods of HUIM are not suitable to deal with such an enormous amount of data, and there is a need for scalable algorithms to mine the itemsets efficiently.

II. RELATED WORK

For mining HUIs, many algorithms have been developed in recent times. Most of these algorithms are for stand-alone systems and small datasets. As the size of the database increases, the performance execution starts to degrade. The computing assets of one machine are not adequate to mine the large datasets and put restrictions on the algorithm's scalability. To overcome these problems, researchers have started to develop algorithms based on the distributed framework. Some of the parallel and scalable utility mining algorithms are being reviewed in this section.

The algorithm PHUI-Growth (2016) is based on Hadoop and proposed by Lin et al. [1]. The algorithm inherits several properties of Hadoop such as scalability, fault tolerance, load balancing, etc. The workload is divided into smaller jobs and various pruning strategies are used to discard the unpromising items locally. But the algorithm suffers from the limitation of multiple data scans. PHUI-Miner (2016) is being proposed by Chen et al. [2]. It is being implemented in Spark as Spark has some advantages over Hadoop, which increases the overall efficiency of the algorithm. This method is a tradeoff

between accuracy and enactment as the mining strategy is based on compression and sampling, which provides approximate HUIs. Still, this method performs better with sampling and lowers the computation time and memory usage. BigHUSP (2016) is another Spark-based algorithm for big data, which uses several properties of Spark and implements the MapReduce framework for mining [3]. A novel data structure called utility matrix is used to store the utility data of the intermediate candidates. Ashish Tamrakar [4] has proposed EFIM-Par for large data sets (2017), a parallel implementation of the most efficient algorithm for smaller datasets, EFIM. A novel strategy of pruning HUI-PR is being developed, based on hash table approach and reduces the search space greatly [5]. The algorithm distributes the search space so that the worker nodes get to work in a well-organized approach. The algorithm performs well with large datasets in comparison to other state-of-the-art algorithms. P-FHM+ is a length constraint algorithm for HUIs, developed by Sethi et al. [6] (2018). It is a parallel modification of FHM+ and outperforms it in terms of scalability. But the algorithm lacks the feature of load balancing for the worker nodes. Sethi et al. has proposed another parallel variant of HAUI-Miner called PHAUIM [7] where the problem of load balancing has been resolved by equally sharing the workload among the nodes (2019). A superior technique of search space division has been established; a new upper bound method called as average utility upper bound is presented, which in combination with the dimension of search space generates an inducing factor for nodes to equalize the load evenly among them. Nguyen et al. [8] has proposed another parallel modification of EFIM, called pEFIM (2018). This algorithm uses multi-processor, shared memory-based architecture for parallelism. The original EFIM uses a depth-first approach and so the method pEFIM perfectly adopts the strategy for the parallel execution. Although pEFIM uses more memory as the threads in the algorithm have their own private space. The speed of the algorithm increases with an increase in number of threads.

The demand for mining the utility items has grown over the past years, as utility mining is an important task for many application areas such as market-basket analysis, e-commerce, biomedicine, etc. But most of the algorithms [9], [10], [11], [12], [13], [14] are not competent enough for the large datasets and do not scale well with the growing data. A novel technique has been recommended by the authors Dalal et al. [15] for mining high utility itemsets from large datasets, named as Absolute High Utility Itemset Mining or AHUIM algorithm. The proficiency of the algorithm is being evaluated with other state-of-the-art algorithms in this research work.

A. Preliminaries and Problem Statement

The perception of HUIM was first presented by Yao et al. [16]. It is briefly summarized in this section. Consider I as a set of finite 'g' items, I = {I_1, I_2, I_3…I_g} for a transactional dataset DS with set of transactions $T_S \in$ DS. Transactions have a distinctive identifier, called as Transaction ID (T_{ID}), $1 \leq s \leq n$. Let I be an itemset of n-items, called as n-itemset. Table 1 shows an example dataset with five transactions and seven unique items. Table 2 displays the profit value of each item.

Table 1: Transactional Dataset DS

Tid	Transactions (item:quantity)	Transaction-Utility
[1]	(L:2), (N:3), (O:2)	24
[2]	(L:3), (N:2), (P:4), (R:2)	37
[3]	(L:2), (M:3), (O:2), (P:1), (Q:5)	40
[4]	(M:3), (O:5), (P:2)	52
[5]	(M:4), (N:1), (P:4), (Q:6)	42

Table 2: Item-Profit Table

Item	L	M	N	O	P	Q	R
Profit value	3	4	2	6	5	1	2

Some of the technical notations can be described as follows-

For of an item I_k in a transaction T_s, Utility; is represented as $U(I_k, T_s)$ and it can be expressed as the multiplication of internal-external utility of the item. Internal Utility = quantity of item I_k in the transaction T_s denoted as $q(I_k, T_s)$ and external utility = profit value of the item, denoted as $p(I_k)$. So, $U(I_k, T_s) = q(I_k, T_s)* p(I_k)$. From the dataset of table 1, utility for the items L, N and O in transaction T_1 is 2*3 = 6, 3*2 = 6 and 2*6 = 12 correspondingly.

For an itemset I in a transaction T_s, Utility; can be expressed as $U(I, T_S) = \sum U(I_k, T_s)$ for $I_k \in I$. Utility of an itemset {L, N} in T_2 is denoted as $U(\{L, N\}, T_2) = U(L, T_2) + U(N, T_2) = 3*3 + 2*2 = 13$.

Transaction Utility; for a transaction T_s, TU is stated as the integration of utility of each of the item for that transaction. For example, TU for T_1 = 6+6+12 = 24

Transaction Weighted Utility; TWU of an itemset I is stated as summation of utility of every transaction containing itemset I. For example, TWU of 1-itemset {L}= Summation of transaction utilities of T_1, T_2 and T_3 = 24+37+40 = 101. Table 3 shows the TWU value for 1-itemsets of the dataset.

High Utility Itemset; an itemset I is categorized as high utility itemset (HUI) iff the utility of itemset I is no less than the minimum threshold, Th-util else, I is an itemset of low utility.

HUI = {I | U(I) ≥ Th-util}

Table 3: 1-TWU items

Itemset	{L}	{M}	{N}	{O}	{P}	{Q}	{R}
TWU	101	134	103	116	171	82	37

B. Problem Declaration

Given a dataset 'DS' and pre-specified threshold 'Th-util', the aim is to uncover all the high utility itemsets in the distributed environment by parallel mining of the dataset over numerous nodes.

III. PROPOSED TECHNIQUE – AHUIM

The authors Dalal et al. [15] have projected a novel technique called Absolute High Utility Itemset Mining (AHUIM) algorithm. The technique extends the basic EFIM algorithm with the divide and conquer tactic to manage the large datasets effectively. AHUIM simulates the mining process on the parallel framework of Apache Spark, which is an open-source cluster-computing platform. Spark has one main driver/master program to execute the main function and multiple other nodes to run the parallel tasks. It has many inbuilt features such as fault tolerance, scalability and in-memory computation. The dataset is stored in RDD format (Resilient Distributed Dataset) [18], [24] inside Spark that is a read-only collection of items and partitioned through various nodes. Spark also has a view of shared variables.

The algorithm AHUIM initially creates a utility list for the items of the database. The utility for an item is comprised of internal and external utility. The said utility list is used to calculate the transaction-weighted utility (TWU) of all 1-itemsets. The 1-itemsets, with values less than the pre quantified threshold value, Th-util, are discarded here only as the unpromising items. The rest of the items (with TWU values more than or equal to Th-util) are organized in the arising order of their TWU values. After the deletion of unpromising items, if there is any null transaction in the dataset, the transaction is clipped off. This dataset is termed as the revised dataset. The sorted items are organized in an enumerated tree configuration, which is then shared with the existing nodes of the cluster. Apart from TWU, two other strategies are used to prune the items that sound unpromising. The first is absolute local utility, and the other is absolute subtree utility. These strategies provide tighter upper bounds and efficiently reduce the search space, thus saving the computation resources from needless traversing and processing. The technique used to split the search space is exemplified in the figure below.

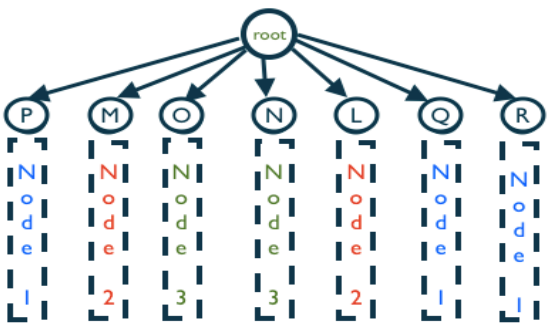

Figure 1 Division of Search Space Among the Nodes

Items are allocated to the nodes starting from the first node and then from the last node to balance the load. This process keeps on iterating until all the items are allocated to the nodes, called worker nodes. These nodes are responsible for mining the HUIs for their search space. For example from figure 1, node 1 is responsible for sourcing the high utility itemsets from the search space of items P, Q, and R. The whole process runs recursively in parallel using the MapReduce structure of the Spark framework. The flow graph of the algorithm is being represented in the figure 2.

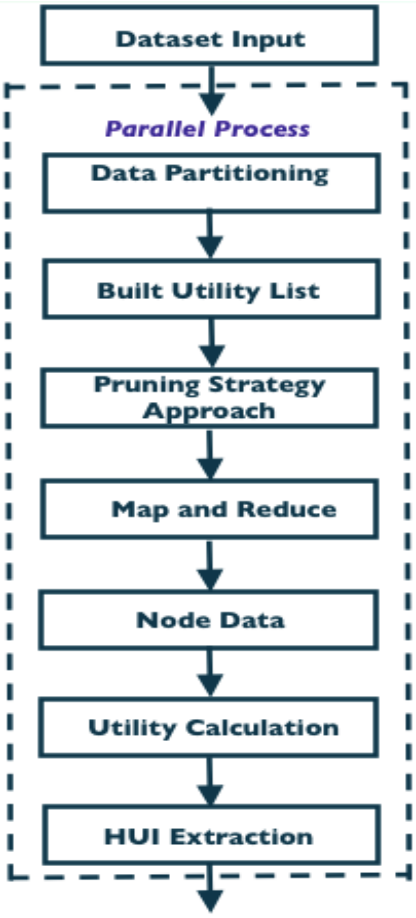

Figure 2 Flow Graph of AHUIM

IV. EXPERIMENTAL RESULTS

The technique AHUIM is evaluated in this section. To evaluate the functioning, it is being competed with two state-of-the-art algorithms of high utility itemset mining for big data– EFIM-Par and PHUI-Miner. These algorithms, along with the AHUIM algorithm have been implemented in Python using IDE Spyder4. A Spark cluster with one master node and six working nodes has been constructed for the distributed framework using Apache Spark 3.0. The system used for execution has 32 GB RAM with 2 processors x Intel® Xeon® CPU E5-2620, 6 cores per processor @ 2.00 GHz. The operating system is Windows 10.

A. Dataset

The experiments are conducted on real-world datasets-Chess, Connect, and Mushroom. Chess is the dataset for different movement of games with 3196 transactions and 75 distinct items. Connect is also a dataset for game with

67557 transactions and 129 different items. Mushroom is a dense datasets for various mushroom's varieties with 8416 number of transactions and 119 unique items. These datasets are freely available in the UCI repository with detailed information about various attributes [22], [25]. The datasets are being replicated with scalar factors to increase the size as shown in table 4. All the experiments are conducted 5 times and the average values are being taken.

Table 4: Various Datasets for Experiments (Source-UCI Repository)

Dataset	#Transactions	#Items	#Average Items
Chess30x	95880	75	37
Connect2x	135114	129	43
Mushroom20x	168320	119	23

B. Performance Evaluation

To compare the performance of the proposed AHUIM algorithm, it is being compared with EFIM-Par and PHUI-Miner for execution time, scalability, memory usages and accuracy.

a) Time Efficiency: Table 5 shows the execution time for all the three algorithms with different utility thresholds on the datasets Chess30x, Connect2x and Mushroom20x. As shown in the figures, the algorithm AHUIM performs better than EFIM-Par and PHUI-Miner for all user specified threshold values. For example, for the dataset Mushroom20x, the execution time is 47.506 seconds for EFIM-Par and 49.376 seconds for HUI-Miner. In contrast, the algorithm AHUIM takes 46.952 seconds for the user specified threshold value of 5. Similarly AHUIM runs faster with other values of thresholds also, as shown in the figure.

Table 5 Execution Time on Different Datasets

Dataset	Execution Time in Seconds			
	Threshold	EFIM-PAR	PHUI-Miner	AHUIM
Chess30x	4	12.119	13.217	11.907
	5	10.482	12.144	9.716
	6	9.463	10.452	8.047
Connect2x	4	55.475	57.676	51.706
	5	43.432	46.534	43.154
	6	39.943	42.342	37.947
Mushroom 20x	4	52.119	54.617	51.983
	5	47.506	49.376	46.952
	6	39.035	40.519	38.580

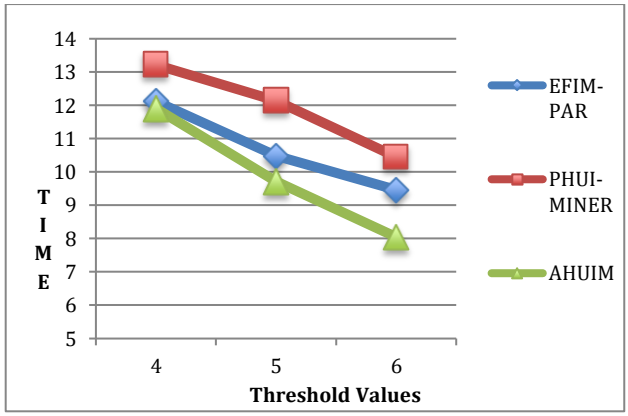

Figure 3 Execution Time in Seconds on Chess30x with Different Threshold

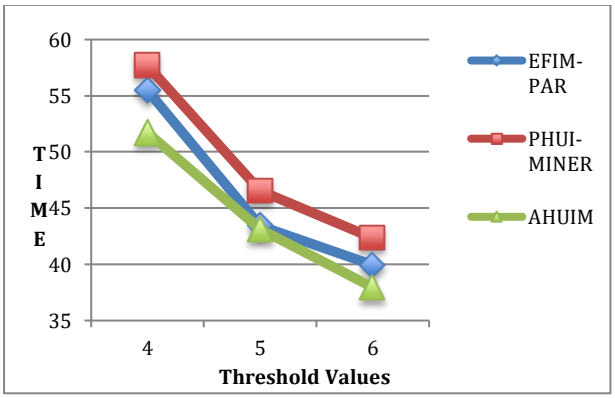

Figure 4 Execution Time in Seconds on Connect2x with Different Threshold

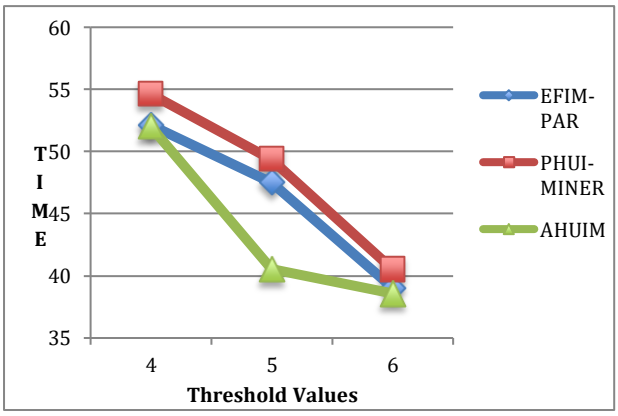

Figure 5 Execution Time in Seconds on Mushroom20x with Different Threshold

b) Scalability: In this section, scalability is investigated for the AHUIM algorithm compared to the other two algorithms. To explore the scalability, two methods are being used.

To see the impact of increase in size of the data

In this method, the dataset is being multiplied by some scalar factor to increase the size of data. For example, dataset Chess is multiplied by a factor of 10, 20 and 30. As seen in figure 6, the running time of the AHUIM algorithm surges slowly and linearly with the increase in the dataset's size (with utility threshold 5).

To see the impact of parallelization

To examine the scalability with the framework and impact of parallelization, the number of working nodes is being changed to 1, 3 and 6. The running time decreases with the increase in number of nodes because of lightening each node's task. This shows the ability of the proposed algorithm AHUIM, to partition the search space well between different machines, which then perform the mining independently and in less time. But the graph is not very linear because of the tradeoff between aids of the parallelism and the state due to inter-communication among the nodes.

Table 6: Execution Time in Seconds (Th=5)

Algorithm	Dataset		
	Chess10x	Chess20x	Chess30x
EFIM-PAR	3.798	7.409	10.482
PHUI-Miner	3.813	6.945	12.144
AHUIM	3.426	5.212	9.716

Table 7: Execution Time for the Algorithms with Different Nodes on Dataset Connect2x (Th=5)

Algorithm	Number of Nodes		
	1	3	6
EFIM-PAR	119.27	97.385	43.432
PHUI-Miner	137.96	123.976	46.534
AHUIM	113.95	91.243	43.154

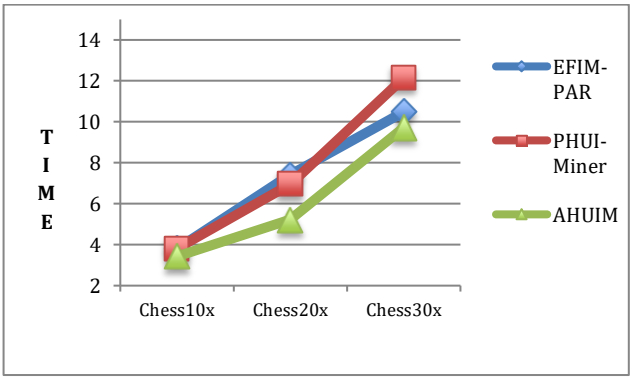

Figure 6 Execution Time in Seconds for Various Sizes of Chess Dataset

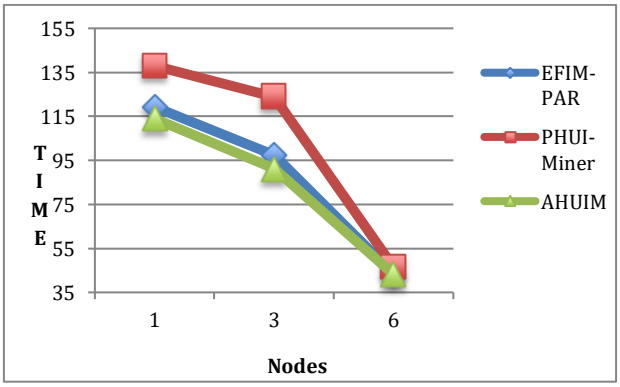

Figure 7 Scalability of different algorithms on dataset Connect2x

***c) Memory Consumption*:** The algorithm AHUIM is compared with the other two algorithms in terms of percentage utilization of main memory (table 8). Figure 8 shows that as the dataset Connect2x is the densest dataset with an average of 43 numbers of items in a transaction, it takes more memory to mine HUIs from it (with utility threshold 5). With the increase in the number of items in a dataset, the data space becomes large. Also, it can be observed that AHUIM takes less memory resources than the other two algorithms because of the pruning strategies ASU and ALU.

Table 8: Memory Utilization by Different Algorithms

Algorithm	Dataset		
	Connect2x	Chess30x	Mushroom20x
EFIM-PAR	0.6428	0.5826	0.6975
PHUI-Miner	0.8935	0.6965	0.8725
AHUIM	0.4638	0.3825	0.4834

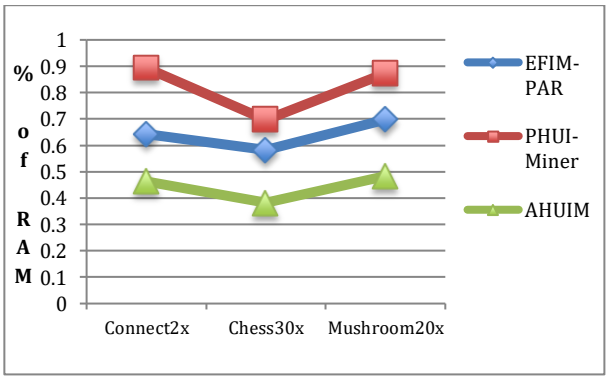

Figure 8: Memory Utilization by Different Algorithms

d) Accuracy: The accuracy for the itemset/pattern generation algorithms can be found by comparing the number of itemsets generated by the algorithms. By taking one algorithm's output as the ground truth, the output of other algorithms can be compared. Here, the algorithm Two-Phase [20] is considered the ground truth as it does not apply any space-pruning strategy. The algorithm comes as a library function in Python to calculate the utility of itemsets. The number of HUIs generated by the Two-phase algorithm is considered as exact HUIs and the output of other algorithms is taken as calculated HUIs, as shown in table 9. Accuracy has been analyzed by comparing the exact number of HUIs with the calculated number of HUIs by calculating precision and relative utility error, as shown in figures 9 and 10. It has been observed that accuracy of AHUIM is around 94.6% and the algorithm experiences a tradeoff between speed and accuracy for the large datasets.

Table 9 Number of HUIs by Different algorithms (Th=5)

Algorithm	Dataset		
	Chess30x	Connect2x	Mushroom20x
EFIM-PAR	172	233	207
PHUI-Miner	174	232	210
AHUIM	170	228	207
Two-Phase	180	243	212

Precision: It is examined by dividing the calculated number of HUIs of an algorithm by the exact number of HUIs, i.e. Precision = (Calculated #HUIs/Exact #HUIs). Table 10 shows the precision values for the three algorithms.

Table 10: Precision Values for Different Algorithms (Th=5)

Algorithm	Dataset		
	Chess30x	Connect2x	Mushroom20x
EFIM-PAR	0.95	0.95	0.97
PHUI-Miner	0.96	0.95	0.98
AHUIM	0.94	0.93	0.97

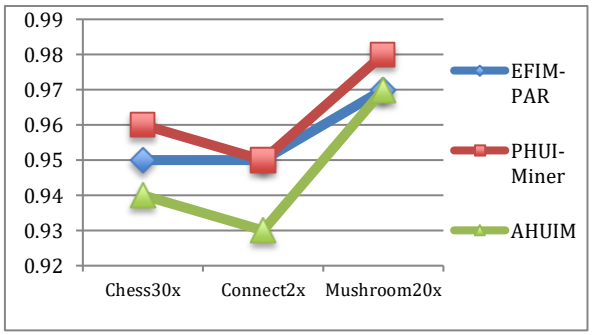

Figure 9: Precision of Different Algorithms (Th=5)

Relative Utility Error: It is examined by the absolute value of difference of HUIs divide by exact number of HUIs, i.e.
Relative Utility Error = abs((Calculated #HUIs- Exact #HUIs)/Exact #HUIs. Table 11 shows the values for the three algorithms.

Table 11: Relative Utility Error for Different Algorithms (Th=4)

Algorithm	Datasets		
	Chess30x	Connect2x	Mushroom20x
EFIM-PAR	0.044	0.041	0.023
PHUI-Miner	0.033	0.045	0.009
AHUIM	0.055	0.061	0.023

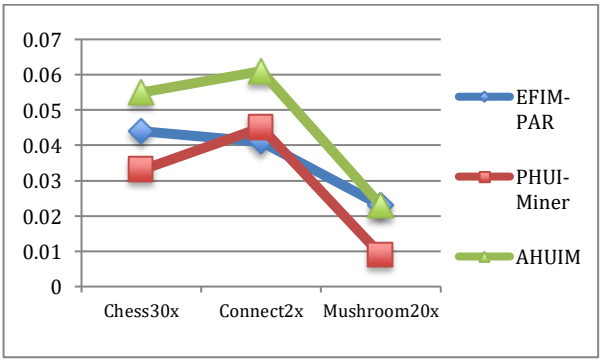

Figure 10: Relative Utility Error (Th=5)

V. CONCLUSION

Conventional data mining algorithms for utility mining are not suitable for big data processing. In this research work, the standalone system is being transacted by distributed system to store and process the big data. A novel approach AHUIM is recommended for mining large datasets in the distributed environment of Apache Spark, which is regarded the most powerful platform for parallel processing. The performance of the algorithm is evaluated with EFIM-Par and PHUI-Miner. The experiments show that the novel algorithm outperforms the other two algorithms for big data in terms of execution time, scalability, and memory consumption.

REFERENCES

[1] Lin, J. C. W., Li, T., Fournier-Viger, P., Hong, T.P., Zhan, J., Voznak, M., "An Efficient Algorithm to Mine High Average-Utility Itemsets", Adv. Eng. Inf. Vol. 30 (2), pp. 233-243, 2016.

[2] Chen, Y., An, A., "Approximate Parallel High Utility Itemset Mining", Big Data Res. 6, pp 26-42, 2016.

[3] Zihayat, M., Hut, Z. Z., an, A., & Hut, Y., "Distributed and Parallel High Utility Sequential Pattern Mining", In 2016 IEEE International Conference on Big Data (Big Data) pp. 853-862. IEEE, 2016.

[4] Tamrakar, A., "High Utility Itemsets Identification in Big Data", Masters Thesis, University of Nevada, Las Vegas, 2017.

[5] Jimmy Ming-Tai Wu, Jerry Chun-Wei Lin, and Ashish Tamrakar, 'High-Utility Itemset Mining with Effective Pruning Strategies', ACM Trans. Knowl. Discov. Data 13, 6, Article 58, 22 pages, 2019.

[6] Sethi, K. K., Ramesh, D. Edla, D.R., "P-FHM+: Parallel High Utility Itemset Mining Algorithm for Big Data Processing", Procedia Compuer Science 132, 918-927, 2018.

[7] Sethi, K. K., Ramesh, D., Sreenu, M., "Parallel High Average-Utility Itemset Mining Using Better Search Space Division Approach", Springer, Cham, pp 233-243, 2019.

[8] Nguyen, T. D., Nguyen, L.T., Vo, B., "A Parallel Algorithm for Mining High Utility Itemsets," Springer, Cham, pp. 286-295, 2018.

[9] Dalal Sandeep, Dahiya Vandna, "Review of High Utility Itemset Mining Algorithms for Big Data," In: Journal of Advanced Research in Dynamical and Control Systems- JARDCS, 10(4), pp: 274-283, 2018.

[10] Vandna Dahiya, Sandeep Dalal, "Big data Mining: Current Status and Future Prospects", International Journal of Advanced science and Technology, Volume 29, No 3, pp. 4659- 4670, 2020.

[11] C. F. Ahmed, S. K. Tanbeer, and B. Jeong, "A novel approach for mining high-utility sequential patterns in sequence databases," In ETRI Journal, vol. 32, pp. 676–686, 2010.

[12] M. Zihayat and A. A. Mining, "Top-k high utility patterns over data streams," In Information Sciences, Available Online, 2014.

[13] Subramanian, K., Kandhasamy, P., Subramanian, S., "A Novel Approach to Extract High Utility Itemsets from Distributed Databases", Computing and Informatics vol 31 (6), pp.1597-1615, 2013.

[14] Zida, S., Fournier-Viger, P., Wu, C.-W., Lin, J.C.-W., Tseng, V.S., "Efficient Mining of High Utility Sequential Rules", In: Proc. 11th Intern. Conf. on Machine Learning and Data Mining, pp. 157–171. Springer, 2016.

[15] Sandeep Dalal, Vandna Dahiya, "A Novel Technique - Absolute High Utility Itemset Mining (AHUIM) Algorithm for Big Data", International Journal of Advanced Trends in Computer Science and Engineering, IJATCSE, Volume 9, Issue 5, pp 7451-7460, 2020.

[16] Yao H, Hamilton HJ, ButzCJ, "A Foundational Approach to Mining itemset Utilities from Databases", In: Proceedings of the 3rd SIAM International conference on data mining, FL, USA, April 2004, pp 482-486.

[17] Borthakur, D., (2007), The Hadoop Distribued File System: Architecture and Design. Hadoop Project Website 11, 21.

[18] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., MaCauley, M., Stoica, I., "Resilient Distributed Datasets: A Fault-Tolerant

abstraction for In-memory Cluster Computing", Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, 2010.

[19] Dahiya Vandna, Sandeep Dalal, "Parallel Approaches of Utility Mining for Big Data", Webology. 17(2), pp – 31-43, 2020.

[20] Liu Y., Liao W., Choudhary A., "A Two Phase Algorithm for Fast Discovery of High Utility Itemsets", Advances in Knowledge and Data Mining, Lecture Notes in Computer Science, Vol 3518, Springer, pp 689-695, 2005.

[21] Sandeep Dalal, Vandna Dahiya, "Big Data Preprocessing: Needs and Methods", International Journal of Engineering Trends and Technology, 68(10), pp- 100-104, 2020.

[22] https://www.philippe-fournier-viger.com/spmf/- An open source Data Mining Library

[23] The Hadoop Project website, [Online]. Available: https://hadoop.apache.org/

[24] The Spark Project website, [Online]. Available: https://spark.apache.org/

[25] The UCI Repository, [Online]. Available: archive.ics.uci.edu/ml/dataset