EFFICIENT MINING OF HIGH UTILITY ITEMSETS FROM TRANSACTIONAL DATABASES

Mr. Vivek J. Jethe[1], Prof. Manoj D. Patil[2], Prof. Sachin Chavan[3]

[1,2,3]*Computer Department, MGMCET, Mumbai university, (India)*

ABSTRACT

In this paper, we would like to show you the working of Hadoop and MapReduce in terms of Data mining. There were many methods proposed earlier for this purpose. Data mining is concerned with analysis of large volumes of data to automatically discover interesting regularities or relationships which in turn leads to better understanding of the underlying processes. The primary goal is to discover hidden patterns, unexpected trends in the data. Data mining activities uses combination of techniques from database technologies, statistics, artificial intelligence and machine learning. Methods like Apriori algorithm, Frequent pattern mining were implemented for this purpose which were not very efficient in terms of time, space and certain parameters like weighted items. These limitations were overcome by bringing into picture the concept of Hadoop and MapReduce.

Keywords : Hadoop, Mapreduce, Data Mining, Utility of Itemsets, Weight

I. INTRODUCTION

To provide the efficient solution to mine the large transactional datasets, recently improved methods are presented in [1]. In [1], authors presented propose two novel algorithms as well as a compact data structure for efficiently discovering high utility itemsets from transactional databases. Experimental results show that UP-Growth and UP-Growth+ outperform other algorithms substantially in terms of execution time. But these algorithms further needs to be extend so that system with less memory will also be able to handle large datasets efficiently. The algorithms presented in [1] are practically implemented with memory 3.5 GB, but if memory size is 2 GB or below, the performance will again degrade in case of time. As when the size of the database increases and if the memory is low, problem arises. We cannot expect the database to be of limited size at least in the real world where commercial aspects are taken into consideration. In the real commercial world, customers are increasing in numbers and their transactions are also increasing and if our database is limited then we are limiting our business which is a shear considerable loss in terms of customer relationship, finance, etc. So to meet such a demand we should be using such a technology that without upgrading the hardware configuration of our systems, we should be able to meet the ever increasing needs of the business growth. Small up gradation would not matter a much for a small business but when it comes to upgrading the whole system, it costs a lot. In this project we are presenting new approach which is extending these algorithms to overcome the limitations using the MapReduce framework on Hadoop.

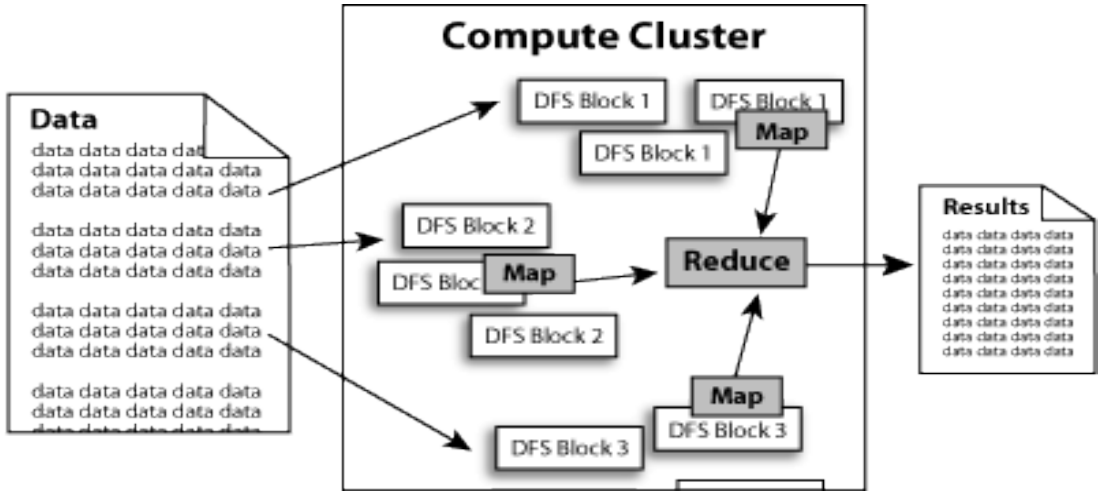
II. UP-GROWTH ALGORITHM

To generate these high utility itemsets mining recently in 2010, UP - Growth (Utility Pattern Growth) algorithm [2] was proposed by Vincent S. Tseng et al. for discovering high utility itemsets and a tree based data structure called UP - Tree (Utility Pattern tree) which efficiently maintains the information of transaction database related to the utility patterns. Four strategies (DGU, DGN, DLU, and DLN) used for efficient construction of UP - Tree and the processing in UP - Growth [11]. By applying these strategies, can not only efficiently decrease the estimated utilities of the potential high utility itemsets (PHUI) but also effectively reduce the number of candidates. But the problem with this algorithm is that this algorithm takes more execution time for phase II (identify local utility itemsets) and I/O cost.

Efficient discovery of frequent itemsets in large datasets is a crucial task of data mining. Lately, several approaches have been proposed for generating high utility patterns; they lead to the problems of producing a large number of candidate itemsets for high utility itemsets and probably degrade mining performance. Mining high utility itemsets from a transactional database refers to the discovery of itemsets with high utility like profits. Though a number of relevant approaches have been proposed in recent years, they lead to the problem of producing a large number of candidate itemsets for high utility itemsets. If there is such a large number of candidate itemsets, it degrades the mining performance in terms of execution time and space requirement. This situation may become worse when the database contains lots of long transactions or long high utility itemsets.

Existing studies applied overestimated methods to facilitate the performance of utility mining. In such methods, potential high utility itemsets (PHUIs) are found first, and thus an additional database scan is performed for utilities' identification. However, existing methods often generate a huge set of PHUIs and their mining performance is degraded consequently. This would become worse when databases contain many long transactions or low thresholds are set. The large number of PHUIs forms a challenging problem to the mining performance since the more PHUIs the algorithm generates, the processing time consumed is very high..

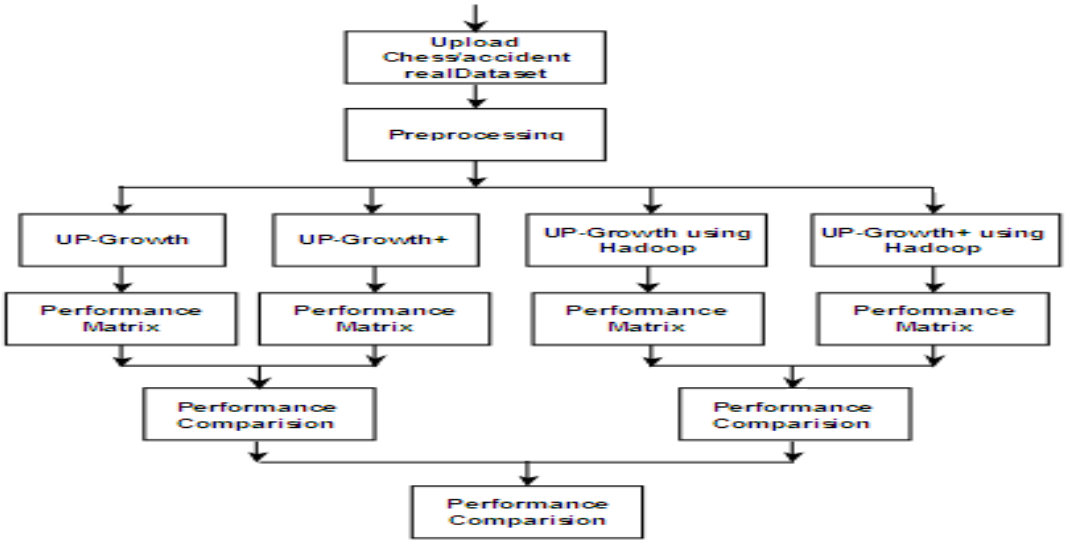
III. OVERVIEW OF HADOOP AND MAP/REDUCE ARCHITECTURE



As shown above, is the architecture of the Hadoop and Map/reduce. There's a heap of data in the first stage as shown in the figure. This data is then processed by the hadoop cluster where we have some mappers and there's a reducer. For each DFS blocks, a mapper is assigned and the result of all the mappers is computed in the reducer. In the reducer stage, the results are computed. So the desired result is achieved.

IV. PERFORMANCE COMPARISON

To understand it better, we will consider a particular dataset as shown in the flowchart. After that, we will apply up-growth and up-growth+ and then calculate the time required. Then apply up-growth and up-growth+ with Hadoop and then calculate its time. Compare the results in terms of time.



V. THE MAPREDUCE FRAMEWORK FOR HANDLING BIG DATASETS

MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. A MapReduce program consists of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each and every name) and a Reduce() procedure that performs a summary operation (such as counting the number of students in each queue, which yields name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates by marshalling the distributed servers, running the various tasks in parallel, which manages all communications and data transfers between the various parts of the system, which provides redundancy and fault tolerance.

"Map" step: The master node takes the input, and then divides it into smaller sub-problems, and distributes them to worker nodes. A worker node repeats this step in turn, leading to a multilevel tree structure. The worker node processes the smaller problem, and passes the answer back to its master node.

"Reduce" step: Master node then collects the answers to all the sub-problems and combines them in some way to form the output – the answer to the problem it was originally trying to solve.

VI. HADOOP OVERVIEW

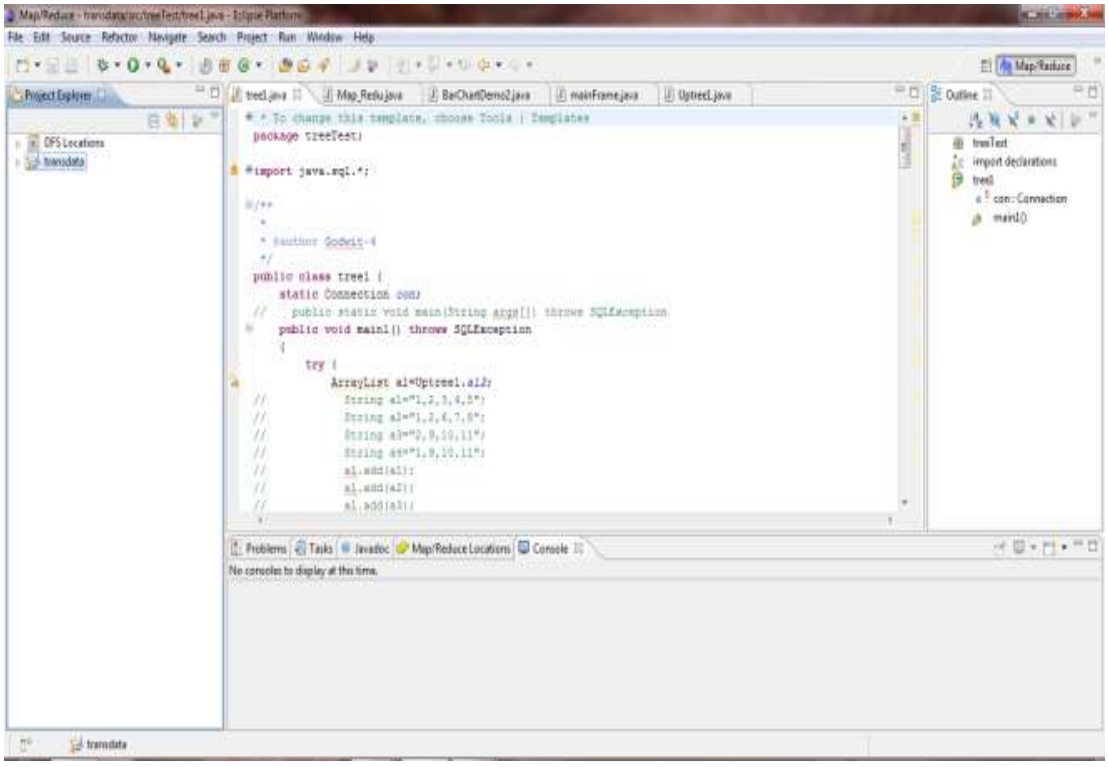
When data sets go beyond a single storage capacity, then distributing them to multiple independent computers becomes important. Trans-computer network storage file management system is called A distributed file system is Trans-computer network storage file management system . A typical Hadoop distributed file system contains thousands of servers, where each server stores partial data of file system.

Hadoop enables a computing solution that is:

- Scalable– New nodes can be added as needed, and added without needing to change data formats, how data is loaded, how jobs are written, or the applications on top.
- Cost effective– Hadoop brings massively parallel computing to commodity servers. The result is a sizeable decrease in the cost per terabyte of storage, which in turn makes it affordable to model all your data.
- Flexible– Hadoop is schema-less, and can absorb any type of data, structured or not, from any number of sources. Data from multiple sources can be joined and aggregated in arbitrary ways enabling deeper analyses than any one system can provide.

Fault tolerant– When you lose a node, the system redirects work to another location of the data and continues processing without missing a beat.

VII. EXPERIMENTAL SETUP



The above screenshot is of the eclipse screen where we have just started and trying to run the code.

The above screen appears when you run the Map/Reduce on eclipse which consists of the code.On this console, we are going to run the test on the datasets with and without Hadoop with Up-growth and Up-Growth+.

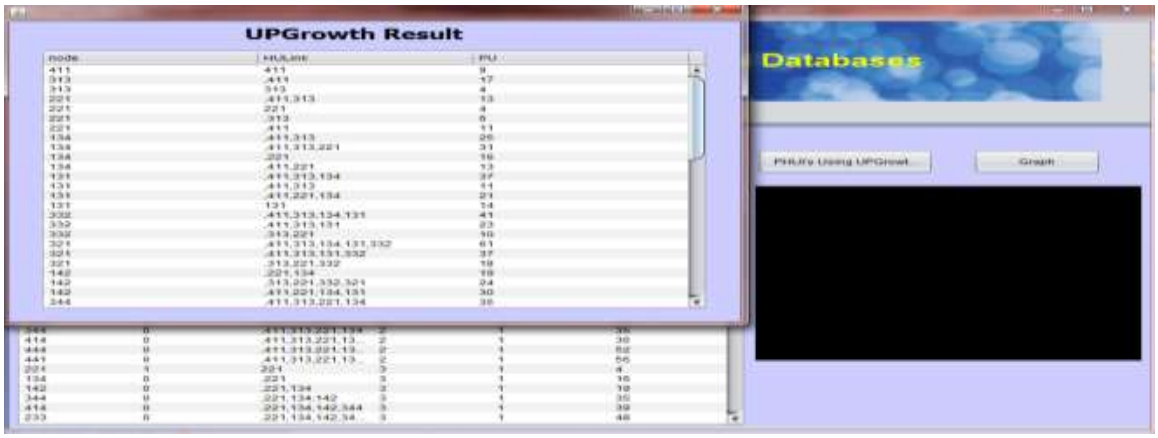
VIII. TESTING WITHOUT HADOOP

The first screen is the loading of the dataset on the console. The first attempt of ours would be to perform data mining without Hadoop and just with Up-Growth and Up-Growth+.

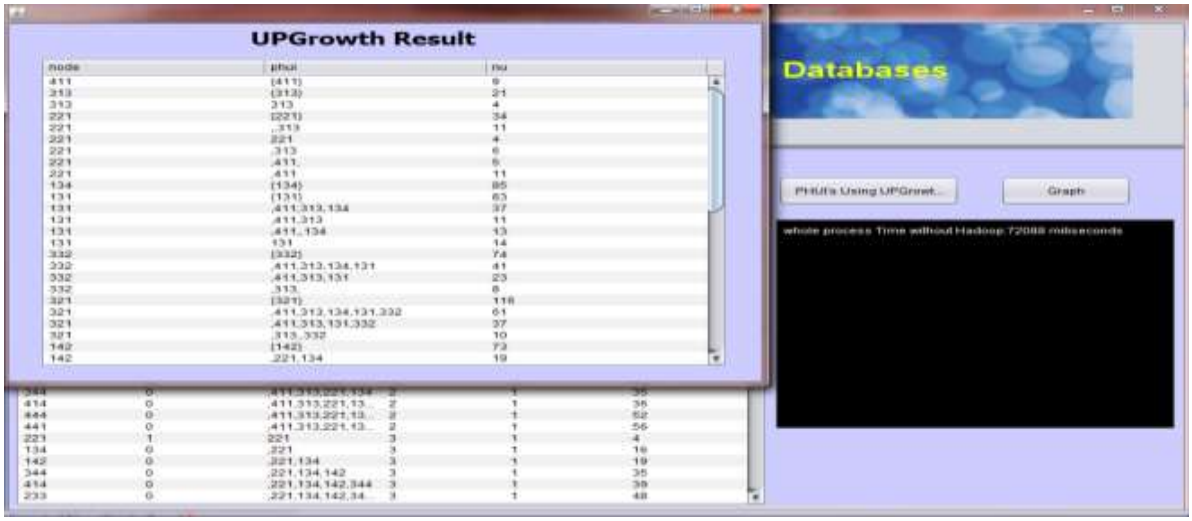
The next step would be to find out the transaction weighted utility by pressing on the respective button on the console. After pressing the button, we get the following output on the screen.

The next step would be to take out the sorted transactions. After clicking on that respective button, we get the following output on the screen.

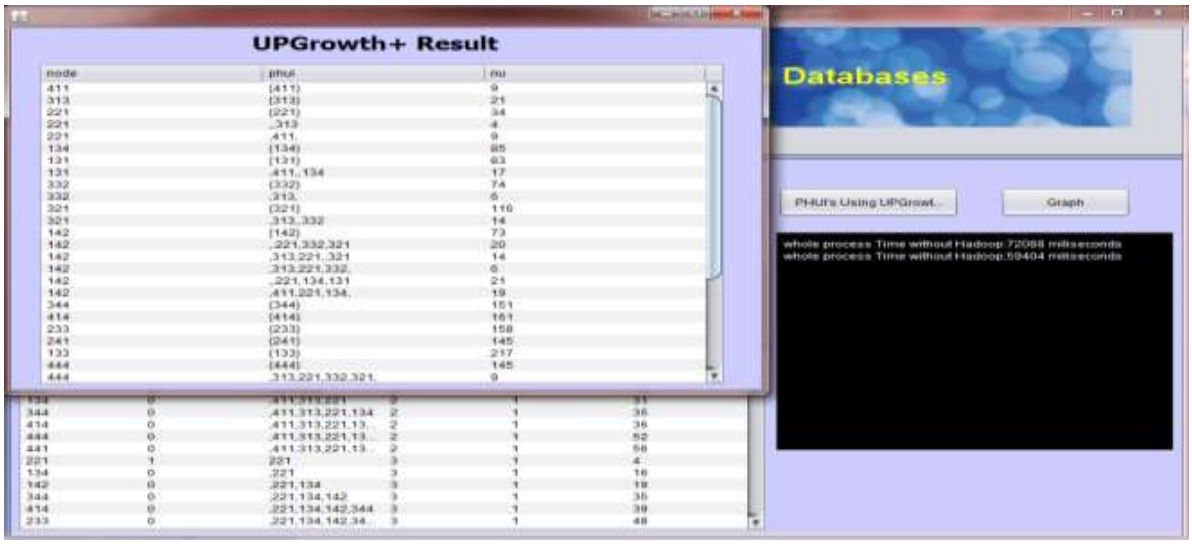
After this, we will generate the Up-Growth tree.



The next step is to find out the number of PHUIs generated and also to calculate the time required for this whole process. The following screen shows,



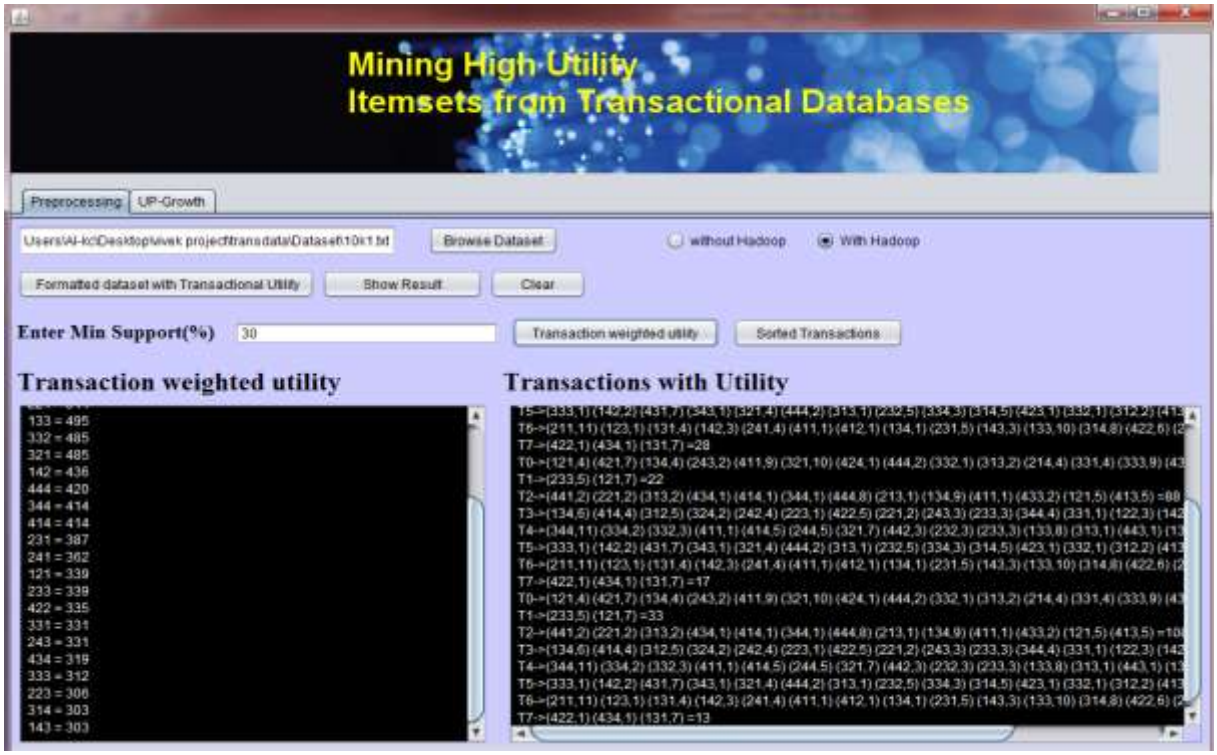
The next step is to calculate the number of PHUIs generated with Up-Growth+ and also to calculate the time required.



In both of the above screens shown, were the output of PHUIs generation with Up-Growth and also Up-growth+, but that was without Hadoop. The time required for Up-Growth is 72088 milliseconds and the time required for the whole process in Up-growth+ is 59404 milliseconds which is less than the previous one. Though the difference is not so high, but it would be of greater concern when it comes to a dataset which is of large size. The time difference would matter a lot and consequently it would also matter to the amount of resources consumed in this process.

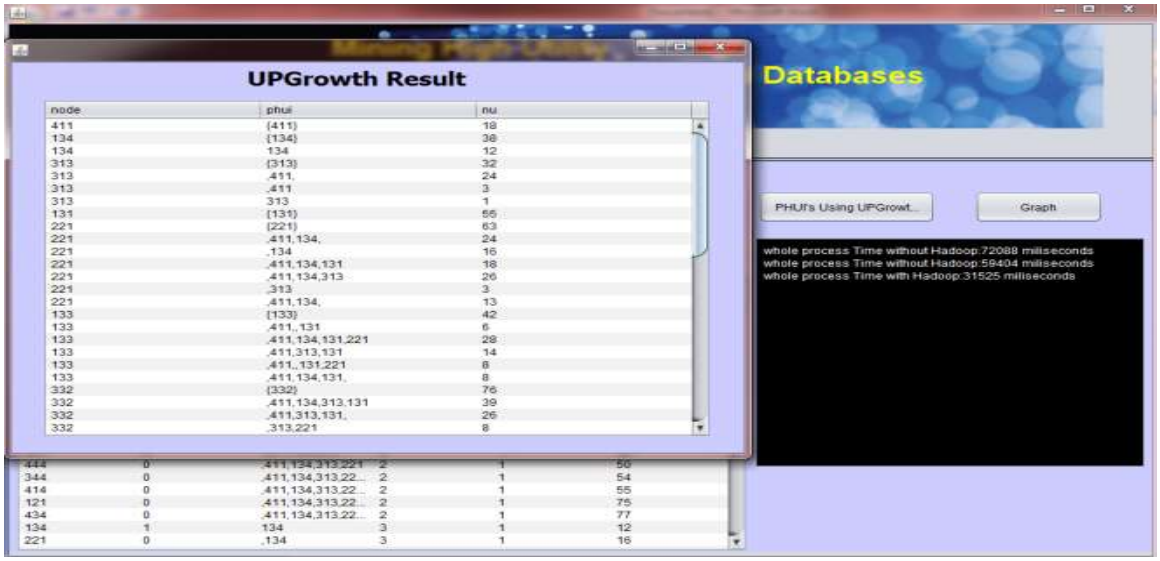
IX. TESTING WITH HADOOP

In this section, we are going to calculate transaction weighted utility with Hadoop.

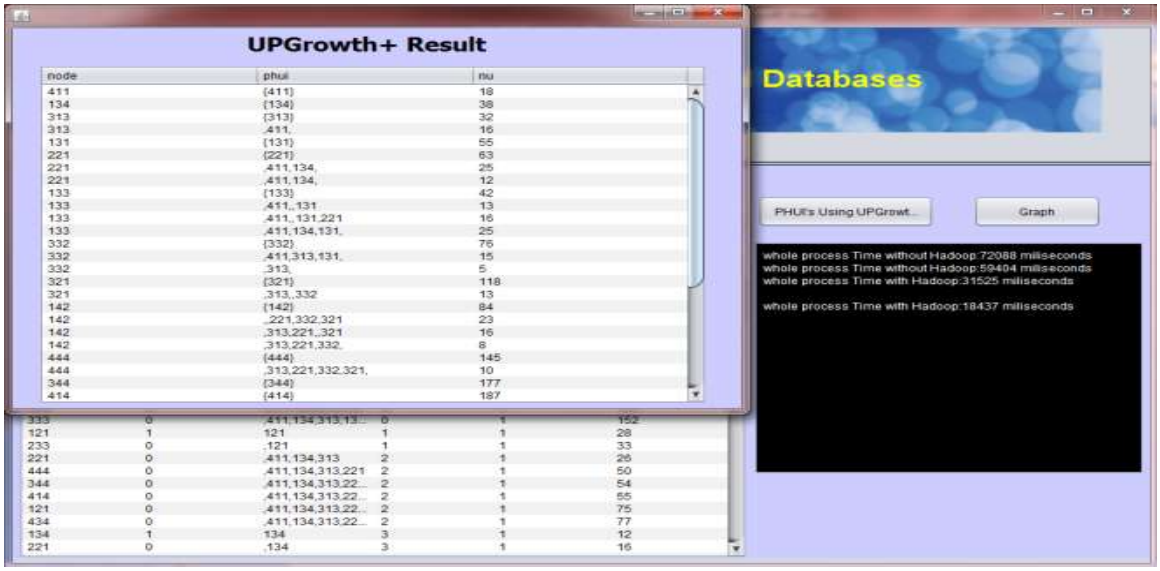


Next step is to find out the sorted transactions.

The next step is to observe the PHUIs generation with UP-Growth which is shown in the screen below.



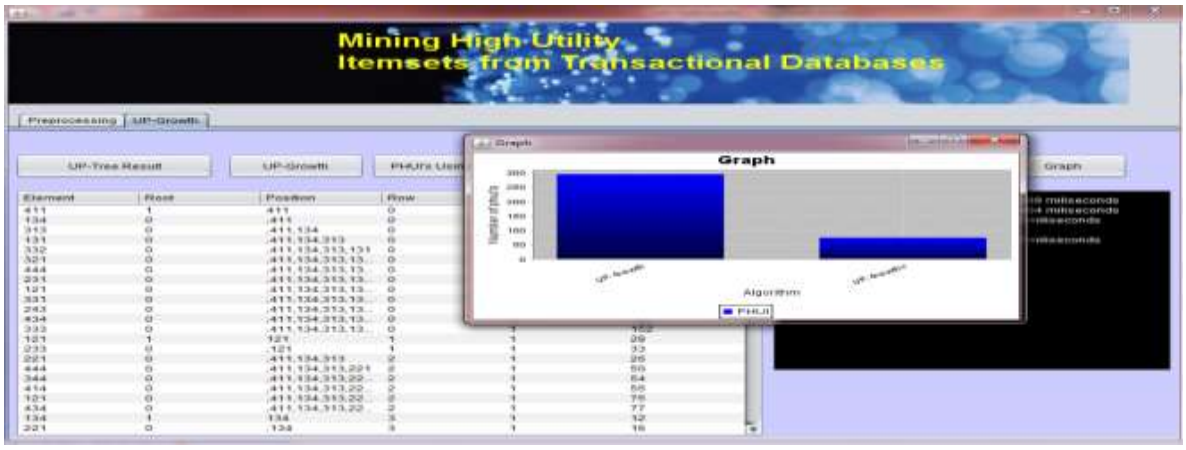
The next step is to observe the PHUIs generation with UP-Growth+.



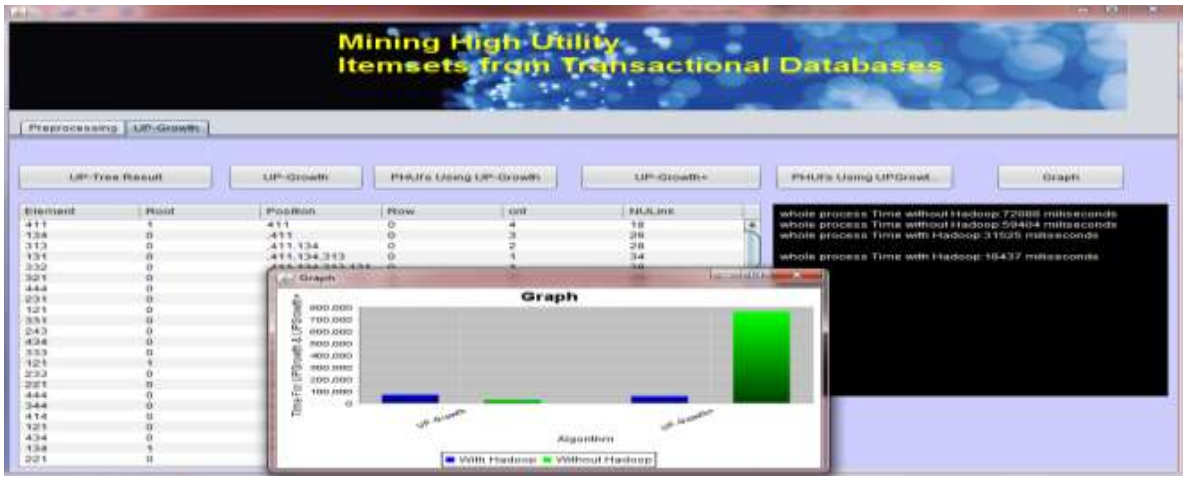
From the above screens, we can see that with hadoop data mining is very efficient as compared to data mining without hadoop. As you can see, the time required for the whole process for Up-Growth with Hadoop is 31525 milliseconds and the time required for the whole process for Up-Growth+ with Hadoop is 18437 milliseconds which is the least.

So we can conclude from these outputs that the time required for the whole process for Up-Growth without Hadoop is the highest whereas the time required for the whole process for Up-Growth+ with Hadoop is the least.

Following screen shows the graph which compares the amount of PHUIs generated in Up-Growth and in Up-Growth+.



Following screen shows the full comparison of the performance i.e. Up-Growth and Up-Growth+ with and without Hadoop.



X. CONCLUSION

As shown above, the time required for the processing of data in Hadoop with UP-Growth+ is very less which proves that it is very efficient as compared to the earlier papers.

REFERENCES

[1] Vincent S. Tseng, Bai-En Shie, Cheng-Wei Wu, and Philip S. Yu, Fellow, IEEE, "Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 25, NO. 8, AUGUST 2013.

[2] V incent . S. Tseng, C. W. Wu, B. E. Shie, and P. S. Yu.: UP – Growth : An Efficient Algorithm for High Utility Itemset Mining. In Proc. of ACM-KDD, Washington, DC, USA, pp. 253- 262, July 25– 28, 2010.

[3] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," Proc. 20th Int'l Conf. Very Large Data Bases (VLDB), pp. 487-499, 1994.

[4] C.F. Ahmed, S.K. Tanbeer, B.-S. Jeong, and Y.-K. Lee, "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases," IEEE Trans. Knowledge and Data Eng., vol. 21, no. 12, pp. 1708-1721, Dec. 2009.

[5] C.H. Cai, A.W.C. Fu, C.H. Cheng, and W.W. Kwong, "Mining Association Rules with Weighted Items," Proc. Int'l Database Eng. and Applications Symp. (IDEAS '98), pp. 68-77, 1998.

[6] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. ACM-SIGMOD Int'l Conf. Management of Data, pp. 1-12, 2000.

[7] S.C. Lee, J. Paik, J. Ok, I. Song, and U.M. Kim, "Efficient Mining of User Behaviors by Temporal Mobile Access Patterns," Int'l J. Computer Science Security, vol. 7, no. 2, pp. 285-291, 2007.

[8] H.F. Li, H.Y. Huang, Y.C. Chen, Y.J. Liu, and S.Y. Lee, "Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams," Proc. IEEE Eighth Int'l Conf. on Data Mining, pp. 881- 886, 2008.

[9] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated Items Discarding Strategy for Discovering High Utility Itemsets," Data and Knowledge Eng., vol. 64, no. 1, pp. 198-217, Jan. 2008.

[10] C.H. Lin, D.Y. Chiu, Y.H. Wu, and A.L.P. Chen, "Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window," Proc. SIAM Int'l Conf. Data Mining (SDM '05), 2005.

[11] Y. Liu, W. Liao, and A. Choudhary, "A Fast High Utility Itemsets Mining Algorithm," Proc. Utility-Based Data Mining Workshop, 2005.

[12] F. Tao, F. Murtagh, and M. Farid, "Weighted Association Rule Mining Using Weighted Support and Significance Framework," Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD '03), pp. 661-666, 2003.

[13] J. Han and Y. Fu, "Discovery of Multiple-Level Association Rules from Large Databases," Proc. 21th Int'l Conf. Very Large Data Bases, pp. 420-431, Sept. 1995.

[14] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. ACM-SIGMOD Int'l Conf. Management of Data, pp. 1-12, 2000.

[15] S.C. Lee, J. Paik, J. Ok, I. Song, and U.M. Kim, "Efficient Mining of User Behaviors by Temporal Mobile Access Patterns," Int'l J. Computer Science Security, vol. 7, no. 2, pp. 285-291, 2007.

[16] H.F. Li, H.Y. Huang, Y.C. Chen, Y.J. Liu, and S.Y. Lee, "Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams," Proc. IEEE Eighth Int'l Conf. on Data Mining, pp. 881-886, 2008.

[17] Y.-C. Li, J.-S. Yeh, and C.-C. Chang, "Isolated Items Discarding Strategy for Discovering High Utility Itemsets," Data and Knowledge Eng., vol. 64, no. 1, pp. 198-217, Jan. 2008.

[18] C.H. Lin, D.Y. Chiu, Y.H. Wu, and A.L.P. Chen, "Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window," Proc. SIAM Int'l Conf. Data Mining (SDM '05), 2005.

[19] Y. Liu, W. Liao, and A. Choudhary, "A Fast High Utility Itemsets Mining Algorithm," Proc. Utility-Based Data Mining Workshop, 2005.

[20] R. Martinez, N. Pasquier, and C. Pasquier, "GenMiner: Mining nonredundant Association Rules from Integrated Gene Expression Data and Annotations," Bioinformatics, vol. 24, pp. 2643-2644, 2008.