International Journal of Emerging Engineering Research and Technology
Volume 2, Issue 5, August 2014, PP 75-80
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

Maximum Utility Item Sets Using B+ Trees

Singamsetty Anusha

Singamsetty Anusha, NRIIT, Pothavarapadu (Krishna dist), Andhra Pradesh.

Sri G. Karunakar

Associate Professor, M-Tech (CSE), NRIIT, Pothavarapadu (Krishna dist), Andhra Pradesh

Abstract: *Mining high utility item-sets from a transactional database alludes to the disclosure of item-sets with high utility like benefits. Despite the fact that various applicable calculations have been proposed lately, they bring about the issue of creating an extensive number of competitor item-sets for high utility item-sets. The circumstances may get to be more regrettable when the database holds loads of long transactions or long high utility item-sets. In this paper, we propose two calculations, in particular utility example growth(up-Growth) and UP-Growth +, for mining high utility item-sets with a set of successful methods for pruning hopeful item-sets. The data of high utility item-sets is kept up in a tree-based information structure name ductility design tree(up-Tree) such that competitor item-sets could be produced productively with just two sweeps of database. Anyhow utility example may get to and take more cycles for doing productive utility item-set techniques in information set transforming. So we propose to create upgraded result less emphases and afterward subordinate less necessity particular in thing set era. B+ trees were produced in upgraded result. B+ trees don't have information connected with inner part hubs, more keys can fit on a page of memory. In this manner, it will oblige less reserve misses keeping in mind the end goal to get to information that is on a leaf hub.*

Index Terms: *B+Trees, Utility Pattern trees, Design patterns in item-sets.*

1. INTRODUCTION

Information mining is the methodology of uncovering nontrivial, long ago obscure and conceivably helpful data from substantial databases. Uncovering helpful examples covered up in a database assumes a vital part in a few information mining errands, for example, regular example mining, weighted continuous example mining, and high utility example mining. Among them, continuous example mining is a central research point that has been connected to various types of databases, for example, transactional databases, streaming databases, and time arrangement databases, and different application spaces, for example, bioinformatics, Web click-stream examination and portable situations.

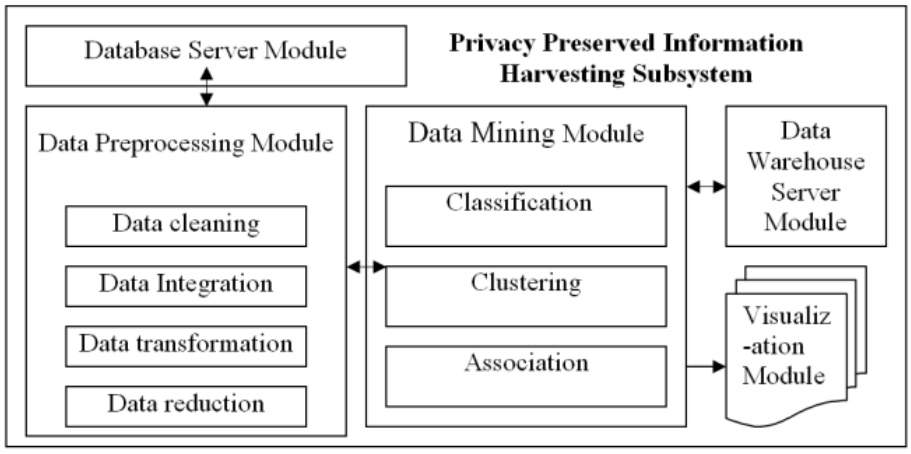

Figure 1. *Data mining operational property with respect to classification and clustering.*

Utility mining develops as an essential theme in information mining field. Mining high utility item sets from databases alludes to discovering the item sets with high benefits. Here, the importance of item set utility is interestingness, imperativeness, or productivity of a thing to clients. Utility of things

in a transaction database comprises of two viewpoints: 1) the imperativeness of unique things, which is called outer utility, and 2) the imperativeness of things in transactions, which is called inward utility. Utility of an item set is characterized as the result of its outer utility and its interior utility. An item set is known as a high utility item set on the off chance that its utility is no less than a client indicated least utility limit; generally, it is known as a low-utility item sets. The productivity of the item sets may achieve excellent and amour nous data set construction which includes effective data representation of the item sets which replies an interactive data procedures in item set construction in finding maximum and minimum utility item set representation for calculating the recent procedures in application process. In this paper we propose to develop an effective mechanism for drawing and intellectual data representation of the item set construction procedure.

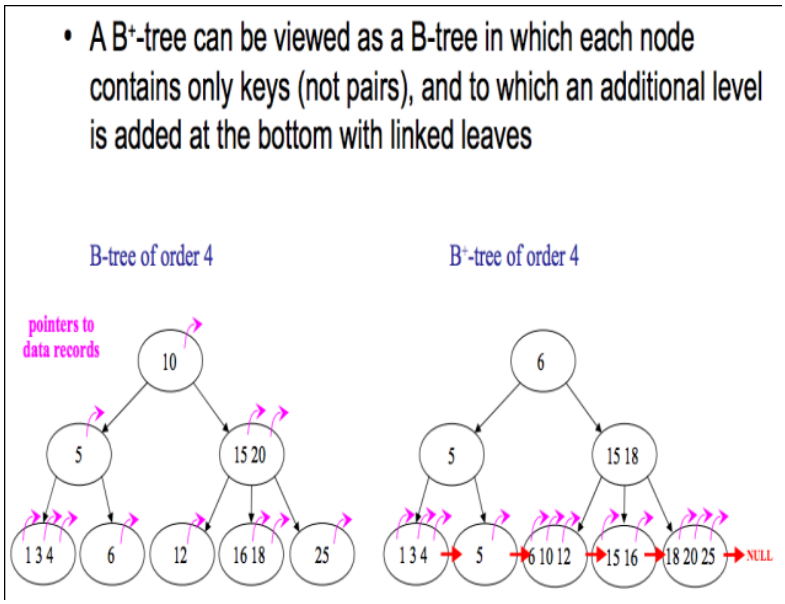

Figure 2. *B+ tree construction procedure for inserting elements in order specification.*

In item set construction using Utility pattern and Utility growth application mechanisms were introduced in data sets construction using Utility pattern tree data structure were used to process individual and combine data sets construction which includes reliable and effective process mechanisms were produced.

Tree traversal is the most mechanism for arranging elements based on number of data sets construction, which defines a relative data representation of the iteration procedures in accessing data procedure. For doing this work effectively we propose to develop B+ trees for arranging item set in equal data presentation. The basic advantage of B+ trees, in that we process data without any associated interior nodes presents in the data sets construction procedure. The leaf nodes of B+ trees are linked, so doing a full scan of all objects in a tree requires just one linear pass through all the leaf nodes. A B tree, on the other hand, would require a traversal of every level in the tree. This full-tree traversal will likely involve more cache misses than the linear traversal of B+ leaves.

2. BACKGROUND WORK

Given a transaction database D and a client determined least utility edge min_util, the issue of mining high utility itemsets from D is to discover the complete set of the itemsets whose utilities are bigger than or equivalent to min_util. In the wake of tending to the definitions about utility mining, we present the transaction-weighted descending conclusion. Albeit weighted affiliation tenet mining considers the vitality of things, in a few applications, for example, transaction databases, things' amounts in transactions are most certainly not taken into contemplations yet. Albeit two-stage calculation lessens pursuit space by utilizing TWDC property, regardless it produces an excess of hopefuls to acquire Htwuis and obliges numerous database checks. To use disconnected things tossing procedure (IIDS) to diminish the number of competitors.

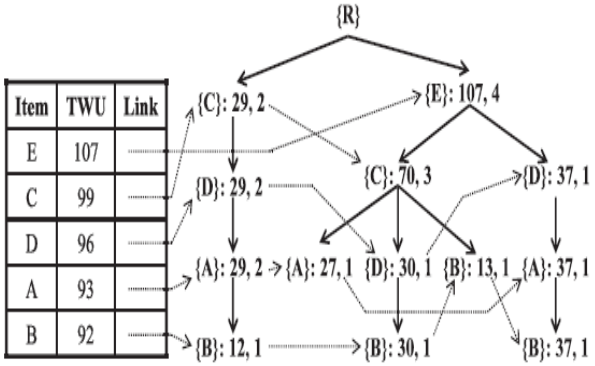

Figure 3. *Data set construction based on the utility item sets*

A treebased structure called IHUP-Tree is utilized to keep up the data about itemsets and their utilities. Every hub of an IHUP-Tree comprises of a thing name, a TWU quality and a help tally. IHUP calculation has three steps: 1) development of IHUP-Tree, 2) era of Htwuis, and 3) ID of high utility itemsets. In step 1, things in transactions are revised in a settled request, for example,lexicographic request, help dropping request or TWU dropping request. To encourage the mining execution and abstain from checking unique database more than once, we utilize a reduced tree structure, named UP-Tree, to keep up the data of transactions and high utility itemsets. Two methods are connected to minimize the overestimated utilities put away in the hubs of worldwide UP-Tree. In taking after areas, the components of UP-Tree are initially characterized. Next, the two methods are presented. At last, how to develop an UP-Tree with the two methods is represented in point of interest by a running case.

3. PROPOSED APPROACH

Although the above calculations serves to attach up the procedure of principle mining by decreasing the aggregate number of database outputs to two, its unpredictability lies in the regular UP tree traversal for getting potential high utility thing sets. Tree Traversal of UP-tree structure obliges different number of emphasess. So we propose to advance the result by utilizing B-tree subsidiary known as B+ tree.

Uses of B+ trees

B+ trees don't have information connected with inner part hubs, more keys can fit on a page of memory. In this manner, it will oblige less reserve misses with a specific end goal to get to information that is on a leaf hub.

The leaf hubs of B+ trees are connected, so doing a full output of all items in a tree requires only one direct pass through all the leaf hubs. A B tree, then again, would oblige a traversal of each level in the tree. This full-tree traversal will probably include more store misses than the straight traversal of B+ takes off.

B trees hold information with each one key, habitually got to hubs can lie closer to the root, and accordingly might be gotten to all the more rapidly. The picture underneath helps demonstrate the contrasts between B+ tree

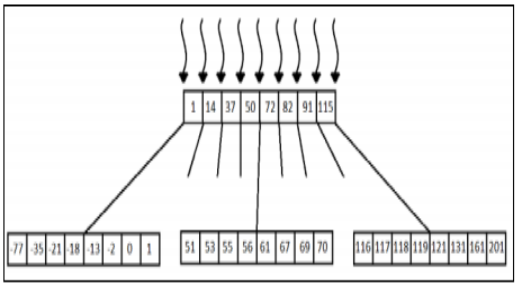

Figure 4. *Example of a B+ Tree with threads checking each key's values against the target value in parallel. In this example, we have 9 threads checking a node of a tree with order 9*

Our introductory codebase is a basic B+ Tree usage by Aviram. Our thinking for utilizing a less complex B+ Tree rather than one utilized within industry is that we are not concerned with subtle elements, for example, locking and memory administration; we are simply intrigued by the hypothetical execution additions to be attained from utilizing the GPU for tree seeks. By utilizing a standalone usage, we can disengage the intrinsic execution of the information structure and evade antiques from additional peculiarities that may be available in a database usage, for example, Sqlite. Beginning with Aviram's straightforward tree, we extended its usefulness to incorporate an extent look and coordinated CUDA piece code for parallel execution of B+ Tree seeks. So as to make a suitable information structure that could be adequately utilized on the GPU, we composed an exchange work that takes the first information structure and makes an equal tree in a solitary coterminous square of memory. This is especially helpful for memory exchanges between the host and gadget (and the other way around) and permits us to abstain from reassigning the pointers to the memory space of the gadget. A B+ Tree seems much like a paired hunt tree, albeit every hub can have up to n-1 keys rather than only two, where n characterized as the request of the B+ Tree. Note that it is not needed for each one key to have n-1 keys; this is simply a most extreme. Every hub has n pointers that indicate up to n kids. Later, we see that picking the request size has an extensive effect on our prosperity with parallel usage. Tree inquiry is a characteristically successive, logarithmic-running time undertaking; the next kid to visit is not known until the question esteem has been checked against the key(s) in the current hub. To influence the GPU's huge number of strings, we utilized an extensive hub request.

4. EXPERIMENTAL EVALUATION

The parameters that we decided to look at for a solitary hunt worth were the number of lines and the measure of the inquiry group. For the reach, we analyzed the impacts of changing the extent of the hunt range. The amount of columns extended from 213 to a greatest of 223 columns. We discovered there to be no distinction in execution whether these columns were ints, glides, and so forth. The amount of inquiries packaged together was shifted from 1 up to 216 – 1, expanding by a component of 4. This upper cutoff is because of CUDA confinements on the measurements of a framework of string squares. As the extent of the group expands, more inquiries may be run in parallel, though the CPU must run these successively. In spite of the fact that the expand in group size implied that there was an build in the memory exchange time for the return set, this time was negligible contrasted with the additional time it took for the CPU to run more questions. Thusly, a bigger group size prompted better GPU execution contrasted with the CPU. In tests where we hold pack size consistent, we picked 10,000 groups as a "preservationist" number to not totally skew execution results when the center is on different variables. Changing the pack size has no immediate impact on the execution of different variables.

We stayed informed regarding timing of the part call as well as the amount time it took to exchange both the starting B+ Tree to the GPU and the exchange time of the results once again to fundamental memory. The exchange time of the B+ Tree structure is critical in this exploration venture, however we expect future work could make utilization of stuck memory to access the B+ Tree from principle memory. We can promptly see the execution picks up from just needing to visit a subset of the aggregate number of hubs in our B+ Tree versus beast energy. In all line sizes indicated here, we see speedups develop from around 10 into the several thousands.

Table 1. Analysis of Single Query Performance

Single Query Performance (100k rows, 128 order)						
GPU time, µs			CPU time, µs	CPU Speedup		
Without Transfer	With Results	All Transfers		Without Transfer	With Results	All Transfers
20	36	1370	2	**10**	**18**	**685**

In any case, when considering exchange times, our speedup stays between 10 what's more 20. This is foreseeable, as the speedup picks up from expanding the amount of lines is bargained by needing to at first exchange over an undeniably bigger B+ tree.

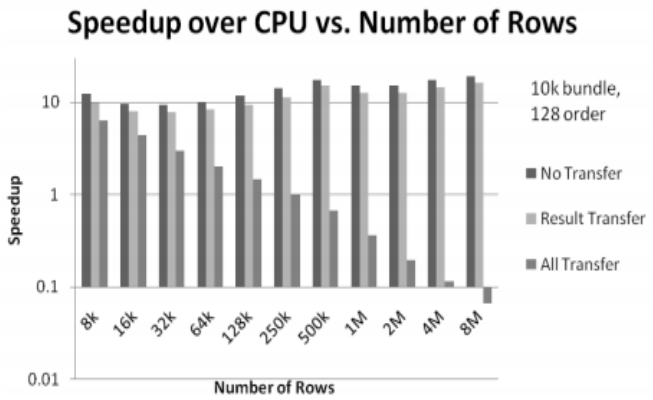

Figure 5. *Speedup over CPU vs. the number of rows with 10k queries bundled and an order of 128. Results are shown for no memory transfer times, result transfer times, and all transfer times included in the speedup*

The B+ tree develops quicker than the show utilized as a part of savage drive as the amount of lines increments; on account of 8m columns, the time for exchange of the B+ tree is 99.7% of the aggregate time for the tree traversal calculation, while the exchange time for beast power is just around 1.2% of the aggregate time. In the mean time, for 8k columns, the tree exchange time represents 48% of the aggregate time of the tree traversal calculation, while the exchange time for beast is just 2.6% of the aggregate time.

The GPU is prevalent in three of the four cases. We see that on the off chance that we overlook exchanges inside and out, the GPU has a speedup of 9.4 to 19.2 over the CPU, with the higher speedups originating from bigger datasets. The all-exchange bar speaks to the speedups for databases obliging continuous overhauls, and the result-exchange bar speaks to the speedups for databases that are rarely overhauled. As should be obvious, the result-exchange speedup is constantly well-over 1, showing that the GPU is quicker for all sizes of rarely redesigned databases. At 250k, we see an equal the initial investment point in the all-exchange bars (converging with 1 on the y-hub), showing that the CPU starts. To beat the GPU on habitually upgraded databases holding more than 250k records. We expect this equal the initial investment point to change relying upon distinctive applications and information sets utilized.

5. CONCLUSION

The aftereffects of this undertaking underscore the force of parallelizing an assignment actually when that errand obliges some synchronization. We see that we pay a cost for keeping the strings synchronized; in any case, the logarithmic number of records that must be gone to (which we can do in parallel) beats this expense, beating serial executions with suitable request, dataset, and pack size for every inquiry. This logarithmic playing point is confirm by the way that the GPU B+ Tree execution beats the GPU animal energy approach on just about all question sets. The results are empowering in that the bigger the dataset size, the bigger the speedup. In view of the results, we contend that utilizing our GPU usage of a B+ Tree is beneficial over serial executions when confronted with a database with occasional redesigns or a little database that has continuous overhauls. This, obviously, is expecting that the capability to plait numerous tree looks together without a moment's delay is plausibility.

REFERENCES

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

[2] J. Pisharath, Y. Liu, B. Ozisikyilmaz, R. Narayanan, W.K. Liao, A.Choudhary, and G. Memik NU-MineBench Version 2.0 Data Setand Technical Report, http://cucis.ece.north western.edu/projects/DMS/MineBench.html, 2012.

[3] B.-E. Shie, H.-F. Hsiao, V., S. Tseng, and P.S. Yu, "Mining HighUtility Mobile Sequential Patterns in Mobile Commerce Environments,"Proc. 16th Int'l Conf. DAtabase Systems for AdvancedApplications (DASFAA '11), vol. 6587/2011, pp. 224-238, 2011.

[4] V.S. Tseng, C.-W. Wu, B.-E. Shie, and P.S. Yu, "UP-Growth: An Efficient Algorithm for High Utility Itemsets Mining," Proc. 16th ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD'10), pp. 253-262, 2010.

[5] P. Bakkum and K. Skadron. Accelerating SQL Database Operations on a GPUwith CUDA. In GPGPU '10: Proceedings of the Third Workshop on General-Purpose Computation on Graphics Processing Units, pages 94-103, New York,NY, USA, 2010. ACM.

[6] H. Yokota, Y. Kanemasa, J. Miyazaki. Fat-Btree: An Update-Conscious Parallel Directory Structure. In Proceedings of the 15th International Conference on Data Engineering (ICDE '99). IEEE Computer Society, Washington, DC, USA.

[7] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander. Relational Query Co-Processing on Graphics Processors. ACM Trans. DatabaseSyst., 34(4):1{39, 2009.}.

[8] B.-E. Shie, H.-F. Hsiao, V., S. Tseng, and P.S. Yu, "Mining HighUtility Mobile Sequential Patterns in Mobile Commerce Environments,"Proc. 16th Int'l Conf. DAtabase Systems for Advanced Applications (DASFAA '11), vol. 6587/2011, pp. 224-238, 2011.