Processing Load Prediction for Parallel FP-growth

Iko PRAMUDIONO†, Katsumi TAKAHASHI†, Anthony K.H. TUNG††, and Masaru KITSUREGAWA†††

† NTT Information Sharing Platform Laboratories, NTT Corporation
Midori-cho 3–9–11, Musashino-shi, Tokyo, 180–8585 Japan
†† Department of Computer Science, National University of Singapore
3 Science Dr 2, Singapore 117543
††† Institute of Industrial Science, The University of Tokyo
Komaba 4–6–1, Meguro-ku, Tokyo, 153–8505 Japan

Abstract Load balancing is a dominant factor to achieve scalable parallel frequent pattern mining. In this paper, we examine some methods to predict processing load for parallel FP-growth algorithm. We propose item processing order based heuristic and load prediction function based on the path depth and other statistics which can be collected before the execution of mining process. We also propose sampling to predict statistics such as the number of iterations. Finally, we implement those methods to improve the initial distribution of processing units i.e. conditional pattern bases as well as the load balancing during the execution of those conditional pattern bases. The performance evaluation shows that sampling based load prediction and item ordering heuristics perform well for the initial distribution.

Key words data mining, parallel-distributed DB, performance evaluation

1. Introduction

Since the invention of Apriori algorithm [2], a lot of algorithms have been devised to cope with the problem of frequent pattern mining. Among them, FP-growth is now well known as a representative algorithm that based on pattern-growth paradigm [5]. The main idea of FP-growth is the projection of the database into a compact on-memory data structure FP-tree. Then it uses a divide-and-conquer method to extract frequent patterns from the FP-tree. FP-growth iteratively construct subdatabases called *conditional pattern bases* for each itemset in the FP-tree.

Some parallel algorithms to mine frequent patterns from a large database also have been proposed because this kind of data mining technique requires a lot of processing power [1], [7], [10], [11], [13].

FP-growth algorithm has some restrictions when implemented on a shared nothing platform, in particular that the tree data structure must stay on the main memory for efficient pointer traversal. The data structure of FP-tree is complex and it is not easy to be partitioned. Thus, the management of memory and work distribution plays important role for the scalability of the system. The first parallel implementation of FP-growth was on a shared-memory machine [13].

In our previous work [10], we have proposed a parallel framework to mine frequent patterns on a PC cluster. The parallel FP-growth algorithm employs the processing of conditional pattern base as the parallel execution unit. We introduced a novel notion of *path depth* to estimate the processing load. Path depth is defined as the longest path in the conditional pattern base whose count satisfies minimum support count. Path depth can be computed from the FP-tree during its construction. Although path depth based load balancing mechanism works well on many datasets, the path depth alone can not measure the workload precisely because the load to process conditional pattern bases depends on some other elements, such as the number of the elements in the conditional pattern bases. In this paper, we focus on how to improve the prediction of processing load. We evaluate some statistics gathered during the execution of parallel FP-growth, and we develop some load prediction methods based on those statistics. We implement the workload prediction during two phases of the parallel FP-growth :

（1） Initial distribution of conditional pattern bases

（2） Load balancing during the processing of conditional pattern bases

The configuration of this paper is as follow : In section 2 we will briefly explain the parallel FP-growth algorithm which becomes the base of this research. In section 3 we will examine some kinds of statistics which can be collected during the execution of parallel FP-growth. Some statistics can be gathered before the processing of conditional pattern base, while we can not know some others until we process them. We further examine how we can utilize those statistics to predict processing load of each conditional pattern base in section 4. In section 5, we will report the parallel FP-growth implementation of those processing load predictions on a PC clus-

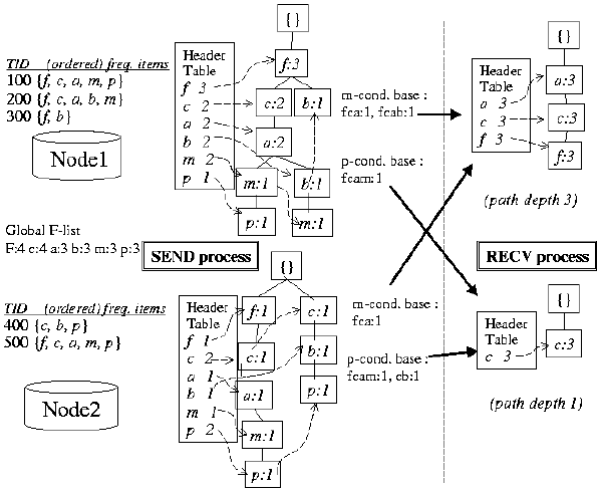

Figure 1 Illustration of parallel FP-growth

ter. Some experiment results using synthetic and real datasets are presented.

2. Parallel FP-growth

First we will explain briefly the parallel FP-growth algorithm. The detail of the algorithm can be found in [10].

The FP-growth algorithm can be divided into two phases : the construction of FP-tree and mining frequent patterns from the FP-tree [5].

2.1 Construction of FP-tree

The construction of FP-tree requires two scans on transaction database. The first scan accumulates the support of each item and then selects items that satisfy minimum support, i.e. frequent 1-itemsets. Those items are sorted in frequency descending order to form F-list. The second scan constructs FP-tree.

As shown in Figure 1, basically we need two kinds of processes : SEND process and RECV process. After the first scan of transaction database, SEND process exchanges the support count of all items to determine globally frequent items. Then each node builds F-list since it also has global support count. Notice that each node will have the identical F-list. At the second database scan, SEND process builds a local FP-tree from the local transaction database with respect to the global F-list.

First, the transactions are reordered according to F-list, while non-frequent items are stripped off. Then reordered transactions are inserted into FP-tree. The order of items is important since in FP-tree itemset with same prefix shares same nodes. If the node corresponds to the items in transaction exists the count of the node is increased, otherwise a new node is generated and the count is set to 1.

FP-tree also has a frequent-item header table that holds head of node-links, that connect nodes of same item in FP-tree. The node-links facilitate item traversal during mining of frequent pattern.

2.2 FP-growth

Input of FP-growth algorithm is FP-tree and the minimum support. To find all frequent patterns whose support are higher than minimum support, FP-growth traverses nodes in the FP-tree starting from the least frequent item in F-list. The node-link originating from each item in the frequent-item header table connects the same item in FP-tree.

While visiting each node, FP-growth also collects the prefix-path of the node, that is the set of items on the path from the node to the root of the tree. FP-growth also stores the count on the node as the count of the prefix path. The prefix paths form the so called *conditional pattern base* of that item.

The conditional pattern base is a small database of patterns which co-occur with the item. Then FP-growth creates small FP-tree from the conditional pattern base called *conditional FP-tree*. The process is recursively iterated until no conditional pattern base can be generated and all frequent patterns that consist the item are discovered.

Parallel FP-growth generates local conditional pattern bases from the local FP-tree. SEND process use hash function to determine which node should process it, instead of processing conditional pattern base locally. Then the RECV process at the destination node collects the local conditional pattern bases from all SEND processes, reconstructs the global conditional pattern base and then executes the FP-growth.

The same iterative process are repeated for other frequent items in the F-list.

3. Some statistics of processing load

Here we examine some statistics related to the processing load of FP-growth. As mentioned in previous section, FP-growth consists of two phases : FP-tree construction and FP-tree traversal. We discuss the processing load of those phases.

Since parallel FP-growth employs conditional pattern base processing as its independent processing unit, we also discuss the statistics specifically related to the conditional pattern base processing. In particular, we will examine how the path depth can be used to predict the processing load of conditional pattern base.

Table 1 gives an illustration of the statistics collected during the mining of frequent patterns. The statistics are taken from fifteen most time consuming conditional pattern bases and they are ranked by the processing time. The dataset is a synthetic T25.I20.D100K.i10K with minimum support 0.1%. The dataset is prepared using the data generator provided by Apriori paper [1]. T25.I20.D100K.i10K means that in the dataset, number of transactions in the dataset is set to 100K with 10K items, and the average transaction size and average maximal potentially frequent itemset size are set to 25 and 20 respectively. Total number of frequent items in the FP-tree is 6689, which means that only 6689 items among 10K items that satisfy the minimum support of 100.

Table 1 Statistics of the most time consuming conditional pattern bases

item order	#iterations	pattern length	#elements	#nodes
5507	16036864	23	107	241
6126	8650752	19	73	297
4792	7920128	22	132	236
4350	5161984	18	132	285
5577	4174752	21	101	236
4164	4072448	21	169	207
4119	2580992	17	155	261
5444	2096640	21	116	286
4066	2083072	21	176	217
3780	1290496	16	168	241
5423	1048576	21	109	253
3526	1041536	20	201	199
4568	659456	16	122	285
4961	524288	20	119	192
3345	524288	20	219	195

3.1 Support of frequent item

During the first scan of transaction data, the support of each item is collected to determine frequent items, called also as 1-itemset. The ranking of the 1-itemsets is represented by the F-list and is used to decide the order of items in the FP-tree.

However as we can observe from Table 1, the order of the items has little relevance with the ranking of the processing load. Among the 6689 items in the F-list, the most time consuming conditional pattern base is the item with the processing order number 5507, 6126 and so on.

The observation also indicates that the ordering of the item processing can affect the overall performance. This optimization will be explored further using the distribution of path depth when we discuss the initial distribution of conditional pattern bases.

3.2 Number of nodes in FP-tree

One of the easily to obtain statistics is the number of nodes in FP-tree. The number of nodes in a FP-tree can be calculated after the construction of the FP-tree. When the number of nodes in a FP-tree is large, more time is needed to traverse it. The total number of FP-tree node traversals is a main factor of FP-growth processing load.

However the number of nodes does not reflect the density of the nodes in the FP-tree. When the nodes in the FP-tree become ”bushy”, a lot of traversals are needed even in the subsequent conditional FP-trees.

3.3 Number of elements in conditional pattern base

Similar to the number of nodes in a FP-tree, the number of elements in a conditional pattern base is also a readily available statistic. The number of elements in a conditional pattern base determines the number of insertions to a FP-tree. The insertion cost is another main factor of processing cost during the construction of FP-tree.

Since each element represents a distinct pattern in the previously generated conditional FP-tree, the number of elements in conditional pattern base also affects the density of the FP-tree, one of the most important performance factor in the tree-based algorithms.

3.4 Number of iterations

Since the processing of conditional pattern bases is the atomic processing unit of parallel FP-growth, the processing load is defined on the processing of each conditional pattern base. And since FP-growth is an iterative process, the processing load is best defined as the number of the iterations during the processing. Note that processing load of a conditional pattern base is also an accumulation of its subordinate conditional pattern bases.

Of course, the exact number of iterations can be determined only after the execution of the conditional pattern bases.

3.5 Length of frequent pattern

The length of frequent pattern is also a good indicator of processing load since the longer the final frequent pattern, FP-growth usually requires more iterations.

Unfortunately, the exact length of frequet patterns also can be determined only after the execution of conditional pattern bases.

3.6 Path depth

The path depth is an approximate of the length of the longest frequent pattern in a conditional pattern base. It can be computed by the bottom-up aggregation of the support counts in the nodes of a FP-tree. When the total aggregated count reaches the minimum support at a certain node, the path depth is the ”depth” of the node, in other word, the number of nodes in the shortest path from the root node to the current node.

Path depth can be calculated during the construction of a FP-tree or a conditional FP-tree. However the accuracy of the path depth is determined from some other factors. The path depth does not represent the actual frequent pattern length because some prefix paths may become infrequent.

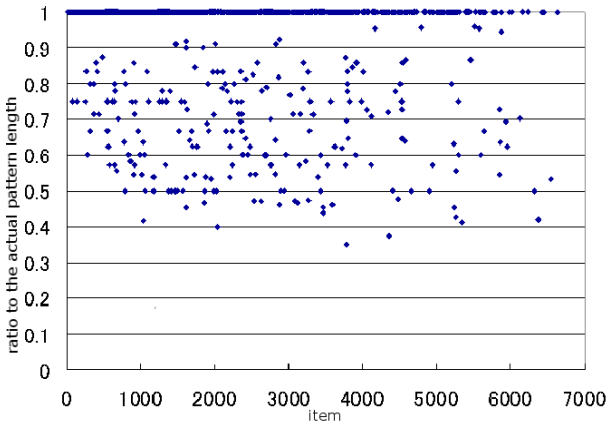

Figure 2 Length of frequent pattern approximation by path depth

Figure 2 shows the accuracy of the path depth to predict the length of the actual longest frequent pattern in a conditional pattern base. The dataset is a synthetic T25.I20.D100K.i10K with minimum support 0.1%. The path depth is calculated from the second generation of conditional pattern bases, i.e. the conditional pattern bases of itemsets with two items.

The X-axis represents the items in the F-list while the Y-axis rep-

resents the accuracy of the path depth which is the path depth divided by the length of the actual longest frequent pattern. The figure indicates that the path depth is a good indicator of the longest frequent patterns after the second generation of conditional pattern bases when the conditional FP-trees are becoming denser. The average accuracy is 0.96 and the standard deviation is 0.12.

4. Load prediction models

Although there are many algorithms proposed so far to mine frequent patterns, there are very few works on quantitative analysis of the mining process. A notion of the "denseness" of a dataset is used to switch data representation during mining process [6], [9]. Some models to predict the processing loads were proposed to select frequent pattern for compression [3] and to switch enumeration method in closed pattern mining [8].

4.1 Load prediction function

When all subsets of a frequent pattern X with length $|X|$ are also frequent and its support is $supp$, the processing load can be estimated as $(2^{|X|} - 1) * supp$ [3].

However if we do not know the length of frequent patterns in advance we need to estimate it. As described in previous section, path depth can be a good indicator of the length of frequent patterns for the second generation of conditional pattern bases or later. Note that the path depth alone is also sufficient to provide a convenient way to balance processing load using a threshold namely *minimum path depth* [10]. Here we explore how we can use path depth to enhance the accuracy of the processing load prediction.

Using the same assumptions as [3], the processing load of a conditional pattern base is proportional to $(2^{\lambda} - 1)$ where λ is the path depth of the conditional pattern base.

The complexity of the processing also depends on the number of elements in a conditional pattern base N_{CPB}, so we examine the load prediction function in the form of $(2^{\lambda} - 1) * N_{CPB}$. Other statistics such as the number of nodes in the FP-tree can be used also, but the number of nodes in FP-tree represents the information of the previous conditional pattern base. We expect to have better prediction since N_{CPB} represents the number of elements in the individual conditional pattern bases generated from the FP-tree.

4.2 Sampling based load prediction

Sampling is one of the earliest method to mine association rule [12]. With some improvements such as lower minimum support threshold, a predictable error rate can be achieved. Some commercial products such as Intelligent Miner also employ sampling to reduce the number of transactions to be processed.

Here we examine sampling not as a method to mine the frequent patterns theirselves, but instead to gather some statistics so we can devise better load balancing strategy. The basic idea is to sample some transactions and then apply mining algorithm on them. From the mining result, some statistics such as the number of iterations, and the length of frequent patterns, which can be observed only after mining finishes, can be estimated.

Because the number of iterations shows the best correlation with the processing load, we use the number of iterations from mining the sample to predict the processing load.

5. Performance Evaluation on PC cluster

A common execution trace of parallel FP-growth is shown in Figure 3. The dataset is T25.I20.D100K.i10K and the minimum support is set to 0.01%. Figure 3 shows the parallel execution without the path depth based load balancing on four processing nodes. The PC cluster consists of 16 processing nodes that are interconnected to a 100Base-TX Ethernet through a Gigabit switch. Each processing node has 800MHz Pentium III and 128 MB of main memory. The underlying parallel FP-growth is implemented in C language on Solaris 8 for x86. The detail of the implementation can be found in [10].

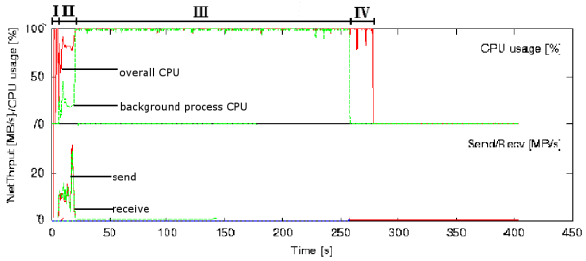

Figure 3 Execution trace of parallel FP-growth without load balancing

The figure shows CPU resource usage, and interconnection network (send/receive). Horizontal axis is elapsed time. The vertical axis for the top graph denotes CPU utilization ratio for overall process (solid red line) and background process (dashed green line). The second graph denotes data transfer throughput in MB/s for interconnection network. The network throughput is divided into two parts, send throughput (solid red line) and receive throughput (dashed green line).

From Figure 3, we can identify that there are two parts where the CPU resource usage is likely to drop thus hinder the scalability of the overall system. The first part (denoted as "II") is when the conditional pattern bases are first extracted from the local FP-tree on each processing node and then distributed to the collector processes. This phase is called "initial distribution of conditional pattern base". The second part (denoted as "IV") is near the end of the execution trace where some processing nodes have starved their own conditional pattern base and ask other nodes to share some conditional pattern bases. Before the processing load is balanced, some processing nodes may become idle. This phase is called "load balancing during the execution of conditional pattern bases".

5.1 Initial distribution of conditional pattern bases

As depicted by Figure 3, during this phase the network activities become the bottleneck of the whole system. The collector process has to collect conditional pattern bases from all nodes before it can

proceed to reconstruct a conditional FP-tree. Actually we can hide the network latency, by employing a background process which can utilyze the CPU idle time to process the reconstructed conditional pattern bases during this critical phase.

However the processing load of each conditional pattern base varies significantly [10]. The background process can not optimally recover the loss of CPU usage if the processing of conditional pattern bases finishes too quickly. Thus it is important to distribute conditional pattern bases that potentially consume a lot of processing time as soon as possible so that the background process has enough processing load to digest.

Unfortunately, as mentioned in Section 3., at this phase only a limited statistics can be gathered because even the conditional pattern bases are not reconstructed. Thus we propose some methods to decide the initial distribution of conditional pattern base using those limited statistics.

5. 1. 1 Distribution methods

Here we examine three kinds of distribution methods. The first one is based on a simple heuristics based on the distribution of path depth to decide which item to start the processing of conditional pattern bases. The rest of the processing is following the order of items in the F-list. The other two methods try to find the optimal distribution by estimating the processing load of individual conditional pattern base. The second method employs the processing load prediction function and the third one employs the sampling based load prediction.

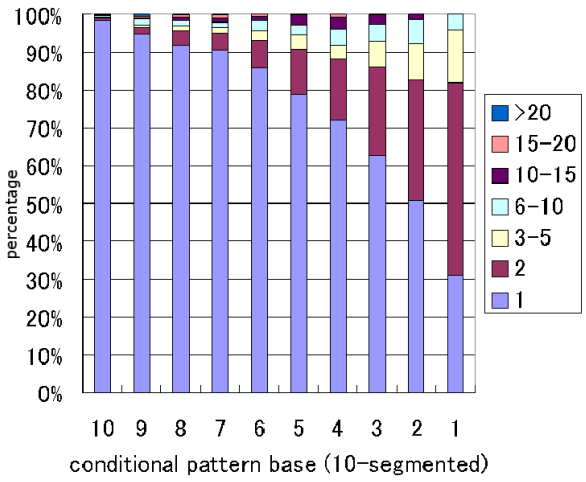

Figure 4 Segmented path depth distribution

（1） ordering of conditional base processing

Figure 4 gives another view of the distribution of path depth. When the items following F-list order are divided into ten equal segments, the vertical axis represents the distribution of path depth for each segment. For example if F-list contains 100 items from 1 to 100, the third segment contains items range from 71 to 80. The color of the bars represents the path depth of conditional pattern bases in the segment. The figure reveals that most of conditional pattern base have small path depth, but some have very large path depth.

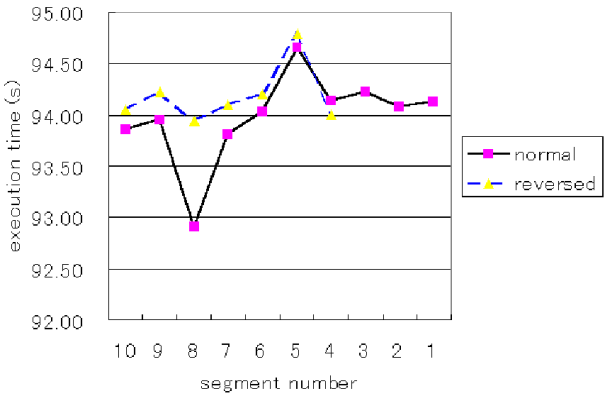

Figure 5 Effect of processing order on performance

Our observation on path depth distribution as shown in Figure 4 also indicates that conditional pattern bases with small path depth are more likely to be found with the segment of the least frequent items in the F-list, i.e. the first segment in the figure. The processing of conditional pattern bases with small path depth is dominated by network activities to send and receive them. Therefore, if following conventional processing order, many nodes stay idle after quickly finishing conditional pattern bases with small path depth. We can make optimization on the processing order.

Figure 5 shows the results of preliminary experiments on the effect of the processing order on overall performance. Four nodes of the PC cluster were used, and the dataset for this experiment was T25.I20.D100K.i10K with minimum support 0.1%. “normal” represents ordinary processing order, i.e. starting from the least frequent item in the F-list to the most frequent, while “reversed” represents the opposite order, i.e. we process the most frequent item in the F-list first. The x-axis is the segment number when the items in the F-list are divided into ten equal segments like in Figure 4. The y-axis records the overall execution time in seconds. The results are the average of two executions.

The figure reveals that the processing order and the starting point for items in F-list have influence on the performance. In particular, the best performance was achieved when following ordinary processing order but starting from third segment, i.e. from 70% to 80% of the items in the F-list. The alternative to reverse the order of the processing does not give significant performance improvement. Nevertheless, as we observe from Figure 4, the first segment also contains mostly conditional pattern bases with small path depth.

Based on our observation, the processing of the items in the F-list is not started from their least frequent ones, but we start from the third segment of the items since in that segment conditional pattern bases with longer path depth are more likely.

（2） Enhanced path depth based load prediction

The prediction of the processing load of the conditional pattern base is based on the processing load prediction function described

Table 2 Execution time of distribution methods (T25.I20.D100K.i10K)

#nodes	ordering	prediction func.	sampling
4	92.9s	103.3s	96.6s
8	52.3s	57.3s	53.4s

Table 3 Execution time of distribution methods (accidents)

#nodes	ordering	prediction func.	sampling
4	193.9s	204.9s	194.8s
8	112.6s	112.6s	96.4s

in previous section.

The statistics from each processing node are collected after the local FP-tree on each processing node is constructed. The aggregated support counts for each ”depth” in the local FP-tree are accumulated to calculate the path depth for each conditional pattern base. The number of elements in each conditional pattern base is also collected so we can calculte the processing load prediction function $(2^{\lambda} - 1) * N_{CPB}$ described in previous section. We sort the predicted load to generate a distribution map determining which processing node should process a conditional pattern base. Conditional pattern bases which are predicted to have more load will be sent first to the processing nodes. Thus we can expect that the background process keep the processing node busy in order to hide the latency during the distribution of later conditional pattern bases.

（3） Sampling

The sampling method collects a portion of transaction database randomly from each processing node and applies the FP-growth mining algorithm to gather the statistics.

The sampled transactions are collected by the control process during the insertions of ordered-transactions into the local FP-tree on each processing node. Those sampled transactions are mined by the control process and the number of iterations for each conditional pattern base is gathered.

The distribution map is generated simply based on the ranking of the number of iterations. The sampling method has the advantage that the number of iterations better represents the processing load. However there are two kinds of overhead during the sampling process : the collection of sampled transactions and the mining on the sampled transactions. Thus we have to set the size of the sampling to get optimal trade off between its precision and the overhead.

5. 1. 2 Performance evaluation of the distribution methods

We perform experiments to compare the effectiveness of each initial distribution method. The datasets are T25.I20.D100K.i10K, which is described in Section 3., with minimum support 0.01% and real data set called ”accidents”. The accidents data set contains anonymized traffic accidents data collected by National Institute of Statistics for the region of Flanders(Belgium) for the period 1991-2000 [4]. It contains 340183 records and 572 items. The average transaction length is 45. The minimum support for accidents dataset is set to 20%.

The results are shown in Table 2 for T25.I20.D100K.i10K dataset and Table 3 for accidents dataset. Those results indicate that changing the order of the distribution shows the best performance because it incurs less overhead. However it is not a trivial task to find the optimal order. Here we use the starting point at the 70% segment in the F-list as a rule of thumb. The sampling based distribution appears to be the second best performer. Here we sample 1% of the transactions. The sampling method also performs better on accidents dataset, in particular when the number of nodes is eight, the sampling method can outperform the item order heuristic based method. The accidents dataset has smaller number of items, and for the minimum support of 20%, the number of frequent items is only 48. Thus the overhead is smaller and the effect of better prediction becomes significant. The sampling method may also work well for some datasets which the characteristics are unknown in advance.

5. 2 Load balancing during the execution of conditional pattern bases

In our previous work we have developed a path depth based load balancing mechanism [10]. By setting the minimum path depth, any conditional pattern base whose path depth is smaller than the threshold will be immediately executed until completion; otherwise, it is executed only until the generation of subsequent conditional pattern bases. Then the generated conditional pattern bases are stored, some of them might be executed at the same node or sent to other idle nodes. Since node with heavy processing load can split the load and disperses it to other nodes, path depth approach can also absorb the processing skew among nodes to some extent.

Here we propose load balancing models which also consider other statistics in order to have better precision of the load prediction. Notice that we can still employ the same load balancing mechanism so that we also need to set a threshold to determine whether to break down the processing load or not.

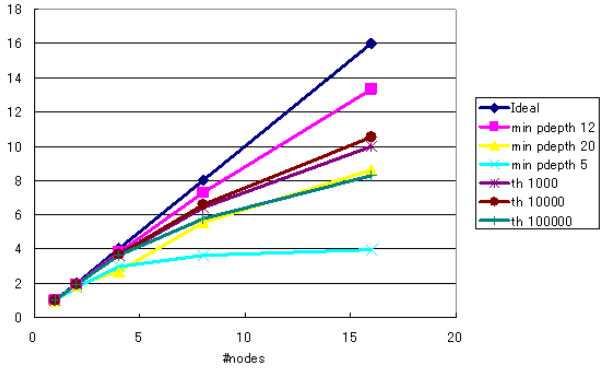

Figure 6 Speedup with T25.I20.D100K.i10K dataset

We employ the load prediction function in section 3.. Figure 6 shows the results of the synthetic T25.I20.D100K.i10K dataset and the minimum support is set to 0.01% for various threshold values. The X-axis represents the number of processing nodes, while the Y-axis represents the speedup ratio, that is how faster the overall

performance when more processing nodes are used. Simple path depth balancing mechanism is denoted by "min pdepth P", where P represents the threshold of the path depth. While the proposed load prediction function is denoted by "th F", where F represents the threshold of the load prediction. For example, "th 1000" means that if the load prediction value of a conditional pattern base is more than 1000 then the subsequent conditional pattern bases will be splitted and stored.

The performance of the load prediction function based load balancing still lacks behind its path depth based predecessor. The experiment results show that the speedup ratio of the load prediction based approach suffer when the number of processing nodes is more than four. The static threshold based mechanism can not sufficiently adapt to the granularity of the processing load distributed among many nodes.

Note also that compared to the simple path depth, it is more difficult to find the optimal value for the threshold because the value range of the threshold is larger. However, as Figure 6 shows, the load prediction function based method is easier to use since the threshold is less sensitive than the minimum path depth.

6. Conclusion

We have examined several statistics related to the processing load of parallel FP-growth algorithm. Based on those observations, we designed two load prediction methods based on path depth and sampling. We have implemented those methods on a PC cluster to examine their effectiveness.

For the initial distribution of conditional pattern bases, we also examined a heuristic to change the processing order based on the distribution of path depth. Through real implementation, we found that the heuristic approach performed better than the load prediction methods because it does not incur overhead to collect the statistics before the distribution. However the sampling method can performs better in some cases where the number of frequent items is small, and it requires less knowledge in advance. We are going to examine other approaches with little overhead.

For the load balancing during the processing of conditional pattern bases, the proposed load prediction function still do not achieve expected performance due to the problem with the flexibility of the static threshold mechanism. The determination of the threshold dynamically during the execution is necessary, although it is not an easy task because more statistics are involved. We are also going to investigate other load prediction models involving more statistics to achieve better prediction precision.

References

[1] R. Agrawal and J. C. Shafer. "Parallel Mining of Association Rules". In *IEEE Transaction on Knowledge and Data Engineering*, Vol. 8, No. 6, pp. 962–969, December, 1996.

[2] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules". In *Proc. of the 20th Int. Conf. on Very Large Data Bases(VLDB)*, pp. 487–499, September 1994.

[3] G. Cong, B. C. Ooi, K.-L. Tan, A. K. H. Tung "Go Green: Recycle and Reuse Frequent Patterns". In *Proc. of Int. Conf. on Data Engineering (ICDE'2004)*, Boston, 2004.

[4] Geurts, K. Wets, G. and Brijs, T. "Profiling high frequency accident locations using association rules". In *Electronic Proc. of the 82th Annual Meeting of the Transportation Research Board*, 2003.

[5] J. Han, J. Pei and Y. Yin "Mining Frequent Pattern without Candidate Generation". In *Proc. of the ACM SIGMOD Conf. on Management of Data*, 2000.

[6] J. Liu, Y. Pan, K. Wang, and J. Han "Mining Frequent Item Sets by Opportunistic Projection" In *Proc. of the ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 2002

[7] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri "Adaptive and Resource-Aware Mining of Frequent Sets". In *Proc. of the Int. Conf. on Data Mining(ICDE)*, 2002.

[8] F. Pang, A. K. H. Tung, G. Cong, X. Xu. "COBBLER: Combining Column and Row Enumeration for Closed Pattern Discovery". In *Proc. of 16th Int. Conf. on Scientific and Statistical Database Management*, 2004

[9] J. Pei, J. Han, H. Lu S. Nishio, S. Tang and D. Yang "H-Mine : Hyper-Structure Mining of Frequent Patterns in Large Databases" In *Proc. of Int. Conf. on Data Mining(ICDM)*, 2001.

[10] I. Pramudiono and M. Kitsuregawa "Tree Structure based Parallel Frequent Pattern Mining on PC Cluster". In *Proc of 14th Int. Conf. on Database and Expert Systems Applications (DEXA'03)*, 2003.

[11] T. Shintani and M. Kitsuregawa "Hash Based Parallel Algorithms for Mining Association Rules". In *IEEE Fourth Int. Conf. on Parallel and Distributed Information Systems*, pp. 19–30, December 1996.

[12] H. Toivonen. "Sampling Large Databases for Association Rules" In *Proc. of the 22th Int. Conf. on Very Large Data Bases(VLDB)*, 1996.

[13] O. R. Zaiane, M. El-Hajj, and P. Lu. "Fast Parallel Association Rule Mining Without Candidacy Generation" In *Proc. of the IEEE 2001 Int. Conf. on Data Mining (ICDM'2001)*, pp. 665–668, 2001