Dynamic Load Balancing for Parallel Association Rule Mining on Heterogeneous PC Cluster System

Masahisa Tamura and Masaru Kitsuregawa

Institute of Industrial Science, The University of Tokyo
7-22-1, Roppongi, Minato-ku, Tokyo 106, Japan
{masahisa,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

The dynamic load balancing strategies for parallel association rule mining are proposed under heterogeneous PC cluster environment. PC cluster is recently regarded as one of the most promising platforms for heavy data intensive applications, such as decision support query processing and data mining. The development period of PC hardware is becoming extremely short, which results in heterogeneous system, where the clock cycle of CPU, the performance/capacity of disk drives, etc are different among component PC's. Heterogeneity is inevitable. Basically, current algorithms assume the homogeneity. Thus if we naively apply them to heterogeneous system, its performance is far below expectation. We need some new methodologies to handle heterogeneity. In this paper, we propose the new dynamic load balancing methods for association rule mining, which works under heterogeneous system. Two strategies, called candidate migration and transaction migration are proposed. Initially first one is invoked. When the load imbalance cannot be resolved with the first method, the second one is employed, which is costly but more effective for strong imbalance. We have implemented them on the PC cluster system with two different types of PCs: one with Pentium Pro, the other one with Pentium II. The experimental results confirm that the proposed approach can very effectively balance the workload among heterogeneous PCs.

Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, 1999.

1 Introduction

Recently commodity based PC cluster system is regarded as one of the most promising platforms for data intensive applications such as decision support query processing and data mining. The power of PC is superior to the workstation for integer performance and the price of PC is also much lower. The floating-point computational power of workstation is higher than PC but usually it is not necessary for database processing. So far extensive researches on parallel database processing algorithms have been done[6]. Most of RDB vendors have developed engines with parallel extensions. Currently parallel execution option is available for most of RDB products. Parallel engine is essential for large-scale data warehouse and is becoming popular nowadays. Thus combining the above two key trends, namely, parallel database processing on PC cluster would be a most cost-effective solution for large scale data warehousing. Many researches on PC clusters are being undergone. However most of them such as Beowulf machines at JPL and Caltech are targeting scientific applications[3]. We have built 100 node PC cluster system named NEDO-100 for data base applications. We implemented parallel RDB kernel on it. TPC-D benchmark and association rule mining were run on the machine[8, 13] and, it showed sufficiently high performance. We exemplified that the PC cluster can achieve considerably improve cost-performance ratio.

The problem we faced in that project is "heterogeneity." The system we built[8, 13] was completely uniform. However, when we planed to increase the number of nodes, it was extremely difficult to find out the same machines. Since the development period of PC is extremely short, configuration of machines is changing so quickly. The clock speed of CPU, the size of main memory, the capacity and transfer rate of disk

drives, all these components are completely different from generation to generation and also different from vendor to vendor . We wanted to increase the number of nodes uniformly, but we could not find out the same PC. Once six months have passed, we have to introduce different type of PCs. Thus heterogeneity is inevitable.

Most of the parallel algorithms developed so far assume the system be uniform. Very few papers address heterogeneity problem[5]. If we apply the parallel algorithm developed for uniform parallel system to the heterogeneous environment, apparently we will see significant performance deterioration. A high performance node can process its allocated task quickly but node with less powerful processor or with low bandwidth disk usually takes longer time to finish. Thus, we have to develop some methodologies to handle these problems, which is the motivation of our research. In near future, the high performance system will be built based on a cluster system, where we will have to face the heterogeneity problem anyway. We picked up data mining as a data intensive application and tried to solve the heterogeneity problem. As the size of database increases, the data mining workload becomes significantly heavy. It is also common that users start from small set of data and gradually increase the size of the data set to be mined. Thus the mining platform can not be uniform but should be heterogeneous PC cluster.

In this paper, we propose run time load balancing algorithms for association rule mining under heterogeneous PC cluster environment. Two strategies named candidate migration and transaction migration are developed. Details on these two will be given in later sections. PCs do not have to communicate each other before the execution in order to normalize the performance among different CPUs and disks etc. During executing data mining, the workload of each node is monitored autonomously and the system performance is controlled to be balanced by migrating candidates/transaction among nodes at runtime.

Section 2 briefly explains the association rule mining and its parallel algorithms. Section 3 introduces the fundamental idea of load balancing for association rule mining. Section 4 describes the detail. Section 5 explains the PC cluster system, and implementation details. Performance evaluation results are given and examined in detail. Section 6 discusses the future work and concludes the paper.

2 Association Rule Mining and Its Parallel Algorithm

2.1 Association Rule

Association rule mining is one of the most well known problems in data mining. Sometime it is also recognized as basket analysis . Its typical application is to find buying pattern in retail databases. An example of an association rule is if a customer buys A and B then 90% of them buy also C . Here 90% is called the *confidence* of the rule. Another measure of a rule is called the *support* of the rule.

Transactions in a retail database usually consist of an identifier and a set of items or itemset. $\{A, B, C\}$ in above example is an itemset. An association rule is an implication of the form $X \Longrightarrow Y$ where X and Y are itemsets, and $X \cap Y = \emptyset$. An itemset X has support s if $s\%$ of transactions contain that itemset, here we denote $s = support(X)$. The support of the rule $X \Longrightarrow Y$ is $support(X \cup Y)$. The *confidence* of that rule can be written as the ratio $support(X \cup Y)/support(X)$.

The problem of mining association rules is to find all the rules that satisfy a user-specified minimum support and minimum confidence, which can be decomposed into two subproblems:

1. Find all combinations of items, called large itemsets, whose support is greater than minimum support.
2. Use the large itemsets to generate the rules.

Since determination of large itemsets from large scale database requires much more processing time, most researches to date have focused on first subproblem. After finding all large itemsets, association rules are derived in straighforward manner.

2.2 Mining Association Rules and Apriori

Here we briefly explain the Apriori algorithm for finding all large itemsets, proposed in [1], since the parallel algorithms we use are based on this algorithm.

In the first pass, support count for each item is incremented by scanning the transaction database. Hereafter we prepare a field named support count for each itemset, which is used to measure how many times the itemset appeared in transactions. Since itemset here contains just single item, each item has a support count field. All items that satisfy the minimum support are picked out. These items are called large 1-itemset. Here k-itemset is defined as a set of k items. The second pass, the 2-itemsets are generated using large 1-itemset that is called the candidate 2-itemsets. Then the support count of the candidate 2-itemsets is incremented by scanning the transaction database. Here support count of the itemset means the number of transactions which contain the itemset. At the end of scanning the transaction data, the large 2-itemsets which satisfy minimum support are determined. The following denotes the k-th iteration, pass k:

1. Generate candidate itemset:
   The candidate k-itemsets are generated using large $(k-1)$-itemsets which were determined in the previous pass. Apriori candidate generation

includes pruning of candidate itemsets that is deleting all of the itemsets in the candidate k-itemset where some of the $(k-1)$-subset of candidate itemsets are not in the large $(k-1)$-itemset.

2. Count support:
   The count support for the candidate k-itemsets are incremented by scanning the transaction database.

3. Determine large itemset:
   The candidate k-itemsets are checked for whether they satisfy the minimum support or not, the large k-itemsets which satisfy the minimum support are determined. The procedure terminates when the large itemset becomes empty. Otherwise $k := k+1$ and goto 1 .

Thus the large itemsets are derived iteratively by scanning the transaction data several times. Apriori is sequential algorithm. In the next section, we examine parallelization methods of Apriori.

2.3 Parallel Association Rule Mining

J.S.Park, et.al proposed bit vector filtering for association rule mining and naive parallelization of Apriori [2, 9], where every node keeps the whole candidate itemsets and scans the database independently. Communication is necessary only at the end of each pass. Local counts are gathered to a certain node at the end of each pass and are summed up to calculate the global count(=support value). Although this method is very simple and communication overhead is very small, memory utilization efficiency is terribly bad. Since all the nodes have the copy of all the candidate itemsets, it wastes memory space a lot.

In [11] Hash Partitioned Apriori(HPA) was proposed. The candidate itemsets are not copied over all the nodes but are partitioned using hash function. The number of itemsets at second pass is usually extremely high, sometimes three orders of magnitude larger than the first pass in a certain retail transaction database which we examined. If the candidate itemsets are partitioned over all nodes' memory space, we can fully utilize the memory of all the nodes. When the user-specified support is low, the candidate itemsets overflow the memory space and incur a lot of disk I/O. By utilizing whole space through partitioning the candidates over nodes instead of duplication, HPA can minimize the extra I/Os.

Hybrid approach between candidate duplication and candidate partitioning is proposed at [7]. The processors are divided into some number of groups. Within each group, all the candidates are duplicated and among groups, candidates are partitioned. [12] proposes the parallel algorithms for mining generalized association rule, which incorporates the classification hierarchy. All the strategies above were proposed for shared nothing parallel machines. Recently parallel strategies for shared memory machines are also proposed[15, 10]. Distributed algorithms are also proposed[4].

However, the algorithms so far proposed assume homogeneous parallel processing environment. In this paper, we propose dynamic load balancing algorithms for heterogeneous parallel systems, where each node might have different type of CPU, and different kinds of disks, etc.

2.4 Hash Partitioned Apriori:HPA

HPA addresses the problem of main memory overflow caused by large number of candidate itemsets by partitioning those itemsets among nodes using hash function as in the hash join[11]. Although it has to exchange transaction data among nodes, its effective utilization of memory space results in better parallelization gain. And using hash function, HPA eliminates broadcasting of all the transaction data and can reduce the comparison workload significantly. In brief, HPA performs following steps:

1. Generation of k-length candidate itemsets:
   At pass k, HPA generates k-length candidate itemsets using large itemsets with length $k-1$ created at previous pass. Then it applies hash function on those itemsets to decide the destination node ID. If the ID is its own, insert it into the hash table. If not, it is discarded.

2. Support counting:
   While reading transaction data, each node generates k-itemsets. Itemsets with support less than user specified minimum support are filtered out while k-itemsets are generated from transaction. Applies the same hash function that used in phase 1 to the k-itemset, and derives the destination node ID and setds the k-itemsets to it. For the itemsets received from the other nodes and those locally generated whose ID equals the node's own ID, search the hash table. If hit, increment its support count.

3. Determination of large itemsets with length k:
   After processing all transaction data, each node determines large itemsets from its own candidate itemsets. Overall large itemsets for pass k are obtained by accumulating large itemsets from all nodes.

In the following sections, we employ HPA as an underlying parallel algorithm.

3 How to balance load for parallel association rule mining

Before describing the detail algorithms in the next section, we will explain the fundamental idea of load

balancing for parallel association rule mining. As described in the previous section of HPA, each node receives the itemsets and probes them against its own hash table. If a node is assigned more candidate itemsets and keeps them as a hash table, it will receive more itemsets from other nodes during counting phase. This means that we can adjust the workload of each node by adjusting the amount of candidate itemsets. If the load of a certain node is higher than the other nodes, we take some of the candidate itemsets from that node and give them to the other nodes. Then the itemsets that are originally directed to that node are now redirected to the new nodes to which the removed itemsets are relocated. Thus the counting workload is migrated from the original node to the other nodes. We name this strategy Candidate Migration. Figure 1 shows the idea.

The workload depends on the itemset. Some itemsets have higher support value, which means those itemsets will receive more counting requests. Thus we have to put weighting factor to each itemset. However real support value is obtained after the execution. So basically the weighting factor is not available before execution. In our approach, we exploit information from the previous pass. Apriori algorithm, as described in section 2.2, consists of several passes. For each pass, transaction database has to be scanned. When we do the load balancing at pass-k, we use the information on the support value of itemset at pass $k-1$. For example, let's consider pass 2, since almost all the time pass 2 is most time consuming. We estimate the support of 2-itemsets at pass 2 using the information on support of 1-itemsets at pass 1. Before the execution of pass 2, the support of 1-itemset is available. We estimate the support of 2-itemset, $support(\{i_1, i_2\})$ by $min(support(\{i_1\}), support(\{i_2\}))$. Here $support(x)$ means the support of itemset x. It can be proved that $support(\{i_1, i_2\}) \leq min(support(\{i_1\}), support(\{i_2\}))$. There might be an error, but much better than not giving any weighting factor. This can be generalized to any pass. Thus we estimate the support of k-itemset by using support of large $(k-1)$-itemsets derived at pass $k-1$ and use it to determine the weighting factor. The details of Candidate Migration algorithm will be given in the next section.

The Candidate Migration is possible if the node still has candidate itemsets to be migrated. The node can transfer the workload by migrating the candidate itemsets. If the skew is high, there arises the case where migrating all the candidates is still not sufficient. In order to handle such situation, we need yet another strategy to migrate workload.

Let's examine the HPA algorithm again in more detail. Each node has two major task. One is to receive the itemset sent from other nodes, probe it against the hash table and increment the count corresponding to that itemset. The other task is to read the transactions from the disk, generate the itemsets and send them to the nodes determined using hash function. We use the former task for Candidate Migration.

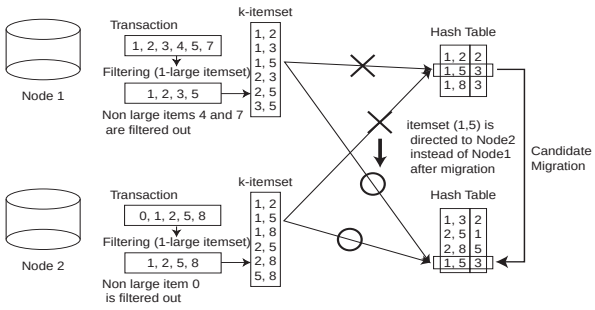


Figure 1: Candidate Migration

Now we consider the latter task. Actually, the itemset generation from transactions is rather complicated process. Transaction contains the items whose support is less than user defined support. So we can eliminate such itemsets by examining the table of large itemsets derived at the previous pass. This workload could be migrated. The node with heavy workload reads the transactions from the disk and it does not do itemset generation itself but just sends the transactions to the light nodes. We name this strategy Transaction Migration.

Transaction Migration incurs overhead of network transfer for each transaction. On the other hand, no additional overhead is incurred for Candidate Migration. Thus, we put priority to the Candidate Migration. Initially heavy node migrates candidate itemsets only. When there are no candidate itemsets remained to migrate, then it migrates transactions. The algorithm to derive the amount of transactions to be migrated will be given in the next section.

4 Run Time Load Balancing Methods

We propose dynamic load balancing methods during the execution of data mining to cope with the skew in heterogeneous system. In this approach, a coordinator node collects necessary information from all the nodes and controls the distribution of workload.

Support counting for pass k in HPA can be divided into 2 processes. The first process is SEND process which comprises reading transaction data from disks, generating k-itemsets and sending the k-itemsets to destination nodes by applying hash function to the itemsets. The second process is called RECV process since processing node receives the k-itemsets from other nodes then probes its hash table to increase the support counts of candidate itemsets. If the time required by node i for SEND process and RECV process are expressed by ST_i and RT_i respectively, overall

CPU processing time for that node can be formulated by:

$$\Delta T_i = \Delta ST_i + \Delta RT_i \tag{1}$$

Here Δ stands for definite time interval.

If we can estimate the required time for each node's SEND and RECV processes to complete the processing of remaining transaction data, we can obtain the estimated remaining processing time for that node as indicated by following expression.

$$\mathrm{rest}T_i = \mathrm{rest}ST_i + \mathrm{rest}RT_i \tag{2}$$

Each of restT_i, restST_i and restRT_i denotes estimated remaining overall CPU processing time, estimated remaining CPU time for the SEND process and estimated remaining CPU time for RECV process respectively.

Since the goal is to have all nodes complete their job at the same time, our method dynamically controls the load allocated for each node so that every node has the same restT_i.

The skew is defined as follow,

$$skew = \frac{\max(\mathrm{rest}T_i) - \min(\mathrm{rest}T_i)}{\mathrm{avg}(\mathrm{rest}T_i)} \tag{3}$$

$$\begin{cases} skew \leq threshold & : no\ skew \\ skew > threshold & : skew\ exists \end{cases} \tag{4}$$

We can judge that the load control is needed if this value exceeds some certain threshold. Here we propose two strategies for balancing the load: Candidate Migration and Transaction Migration.

4.1 Candidate Migration

RECV process can be divided further into 2 subprocesses: (recv1) receives k-itemsets and (recv2) probes the hash table and increment the support count for the corresponding candidate itemsets. If the time for each subprocess are represented by RNT_i and RCT_i respectively, RT_i can be expressed as follow:

$$\Delta RT_i = \Delta RNT_i + \Delta RCT_i \tag{5}$$

Since large scale data mining has to probe large amount of candidate itemsets against hash table, most part of processing time is dominated by (recv2). Thus expression (5) can be reduced to:

$$\Delta RT_i \approx \Delta RCT_i \tag{6}$$

RCT_i itself is proportional to the amount of k-itemsets to receive, thus if we express that amount as RK_i, we can assume:

$$\Delta RCT_i = \alpha_i^1 \Delta RK_i \tag{7}$$

RK_i varies according to the candidate itemsets allocated to that node. RK_i is unknown before execution. But we can estimate RK_i of pass k using the statistics of pass $k-1$ [1]. The support count for a candidate itemset $cand_j = \{t_{j1}, t_{j2}, \ldots, t_{jk}\}$ is always smaller than the least support count of all its subsets[14]. If the support count for an itemset X is defined as S_X, we can define a weighting factor for that candidate itemset as follow:

$$CW_{cand_j} = \min(S_{Lj1}, S_{Lj2}, \ldots, S_{Ljk}) \quad \{L_{j1}, L_{j2}, \ldots, L_{jk}\} \in Sub_j \tag{8}$$

Sub_j denotes a set of all large $(k-1)$-itemsets that are subsets of $cand_j$.

RK_i represents a set of all candidate itemsets in pass k. Then if we represent CD_i as a set of candidate itemsets to be allocated to node i, the weighting factor for that node can be defined as:

$$CV_i = \sum_{\substack{j}}^{|CD_i|} CW_{cand_j} \quad cand_j \in CD_i \tag{9}$$

If overall amount of transactions to be read by all nodes is expressed as DR, RK_i is proportional to the product of CV_i and DR. Therefore, expression (7) can be expanded further into:

$$\Delta RCT_i = \alpha_i^1 \alpha_i^2 CV_i \Delta DR \tag{10}$$

From expression (2) we can express restT_i as:

$$\mathrm{rest}T_i = \mathrm{rest}ST_i + \alpha_i^1 \alpha_i^2 CV_i \mathrm{rest}DR \tag{11}$$

On the other hand, during SEND process each node performs following subprocesses: (send1) read transaction data from database (send2) generate k-itemsets (send3) send them to proper nodes. If the time for each subprocesses are represented by SDT_i, SCT_i, SNT_i respectively, ST_i can be expressed as:

$$\Delta ST_i = \Delta SDT_i + \Delta SCT_i + \Delta SNT_i \tag{12}$$

In most cases of large scale data mining, processing time is dominated by (send2). Therefore, we can approximate expression (12) with:

$$\Delta ST_i \approx \Delta SCT_i \tag{13}$$

Since the load for (send2) subprocess depends on the amount of k-itemsets to send and this amount is proportional to DR_i that is the amount of transactions to be processed by node i, we can express SCT_i as:

$$\Delta SCT_i = \alpha_i^3 \Delta DR_i \tag{14}$$

Hence we have expanded expression (2) into:

$$\mathrm{rest}T_i = \alpha_i^3 \mathrm{rest}SK_i + \alpha_i^1 \alpha_i^2 CV_i \mathrm{rest}DR \tag{15}$$

Here SK_i is denotes the amount of k-itemsets for node i to send out.

[1] See the conclusion on the precision of this approach.

Coefficients α_i^1, α_i^2, α_i^3 are determined using statistical information collected during previous interval. Expression (15) indicates that we can adjust restT_i by varying CV_i.

When *skew* defined in (3), (4) excesses *threshold*, Candidate Migration reallocates candidate itemsets among nodes so restT_i of each node becomes equal. In order to do this, Candidate Migration computes CV_i for all nodes using expression (15) and following restriction: $\sum_i^n CV_i$ = constant We can solve these equations if all CV_is are non-negative, it means there is a solution for candidate migration. Derived CV_i's are used to generate allocation plan for each node, sends the plan and instructs all nodes to begin migration process. Otherwise it sets the negative CV_is to zero before creating allocation plan. Here zero means all the candidate itemsets should be migrated to other nodes. There remains no candidate itemsets. In this case we also need Transaction Migration to be described soon.

As for the implementation, candidate migration requires the remapping of hash table. Once the itemset is migrated, new destination address is put onto the entry so that itemsets are appropriately distributed over the nodes. If remapping is done itemset by itemset, it costs a lot of space. We implemented migration based on range of hash entries.

4.2 Transaction Migration

When load is extremely skewed, we can not rely only on workload migration of RECV process. Thus, we propose another load balancing strategy based on the SEND process.

As described before, in most cases of large scale data mining, processing time is dominated by (send2). Since the load for (send2) subprocess depends on the amount of k-itemsets to be sent, we can express SCT_i as:

$$\Delta SCT_i = \beta_i^1 \Delta SK_i \tag{16}$$

Here SK_i is defined as the amount of k-itemsets for node i to send.

Each node in PC cluster has its own partition of transaction data, Transaction Migration sends some part of that transaction data to other nodes and delegates the generation of k-itemsets to those nodes. In order to do this, each node holds a list of destination nodes and their assignment. This list is dynamically updated during execution. This approach can remove the burden of SEND process of heavy nodes to nodes with excessive computing power, thus it can effectively balances the workload.

As mentioned before, the amount of k-itemsets sent to other nodes SK_i is proportional to the amount of k-itemsets generated by that node. Therefore if node j delegates DM_{ij} $(= -DM_{ji})$ k-itemsets generation to node i and the ratio of DM_{ij} against all the transaction data in node j is defined as $TM_{ij}(= -TM_{ji})$ then SK_i can also be expressed like following:

$$\Delta SK_i = \beta_i^2 (\Delta DR_i + \sum_{j,j\neq i} \Delta DM_{ij}) \tag{17}$$

$$\Delta DM_{ij} = \begin{cases} TM_{ij}\Delta DR_j & TM_{ij} \geq 0 \\ TM_{ij}\Delta DR_i & TM_{ij} < 0 \end{cases} \tag{18}$$

Then the expression (16) will be:

$$\begin{aligned} \Delta SCT_i &= \beta_i^1 \beta_i^2 (\Delta DR_i + \sum_{j,j\neq i} \Delta DM_{ij}) \\ &= \beta_i^1 \beta_i^2 (\Delta DR_i + \sum_{j,j\neq i, TM_{ij}\geq 0} TM_{ij}\Delta DR_j \\ &\quad + \sum_{j,j\neq i, TM_{ij}<0} TM_{ij}\Delta DR_i) \end{aligned} \tag{19}$$

Here coefficients β_i^1, β_i^2 are determined during execution.

Finally we have the restT_i in the following form:

$$\begin{aligned} \text{rest}T_i = \beta_i^1 \beta_i^2 (&\text{rest}DR_i + \sum_{j,j\neq i, TM_{ij}\geq 0} TM_{ij}\text{rest}DR_j \\ &+ \sum_{j,j\neq i, TM_{ij}<0} TM_{ij}\text{rest}DR_i) \\ &+ \alpha_i^1 \alpha_i^2 CV_i \text{rest}DR \end{aligned} \tag{20}$$

From expression (20), we can also control restT_i by delegating k-itemsets generation of SEND process. If Candidate Migration is not enough to overcome the skew, we can supplement it with Transaction Candidate. First, we compute CV_is as described in previous subsection, substitute them into expression (20). For candidate migration, we can analytically derive the solution, but for transaction migration we use hill-climbing method to determine TM_{ij}. TM_{ij} is distributed to the nodes. Nodes with negative TM_{ij} migrate TM_{ij} parts of transactions to light nodes. $(1 - TM_{ij})$ parts of transactions are processed in an ordinary fashion. Nodes with positive TM_{ij} receive the TM_{ij} parts of transactions from heavy nodes.

4.3 Migration plan derivation

Here we assume a coordinator who derives the global migration plan. Coordinator can run on one of the processing nodes or on a separate node. It determines the Candidate Migration and Transaction Migration plan. As mentioned earlier, since the cost of Candidate Migration is smaller, we put priority to Candidate Migration over Transaction Migration. The derivation process is as follows:

1. Acquisition of workload information and skew detection:
   Coordinator acquires workload information from every nodes and computes skew using expression (3). If skew is detected, it proceeds to the following steps.

2. Migration planning:
   Coordinator makes a plan for Candidate Migration using expression (15). If skew still presents, it also creates another plan for Transaction Migration using expression (20).

3. The execution of migration plan:
   Coordinator sends migration plan to all processing nodes and instructs them to reallocate the load. When Candidate Migration is employed, each node transfers candidate itemsets according to the plan and renews the hash table. It also remakes the destination list of transaction data if Transaction Migration is needed.

The above procedure is periodically invoked. Coordinator checks the skew condition every fixed time interval. The complete load balancing is difficult by any means. Error gradually accumulates. Once it becomes beyond the threshold, the coordinator tries to balance the workload again.

5 Performance Evaluation on PC Cluster

5.1 PC Cluster

We have developed a large scale PC cluster consists of 108 PCs interconnected with 155 Mbps ATM and 10 Mbps Ethernet networks[8, 13]. Initially the PC cluster was made up of 100 PCs with 200 MHz Pentium Pro only and then we have added another 8 nodes but with more powerful 333 MHz Pentium II since the performance of PC hardware had improved dramatically. We implement our load balancing strategies on this heterogeneous system. We create two processes on each node, one handles SEND process and the other takes care of RECV process.

5.2 Experimental Environment and Transaction Dataset

In order to simplify the problem and to show clearly the effectiveness of our approach we have made performance evaluation on a group of four nodes each with different CPU power, disk performance and data distribution as shown in table 1. The datasets that mimic retail sales data are generated using procedure described in [1]. The parameters used are described in table 2. Dataset 1 and dataset 2 have 1 and 1.5 million transactions respectively. In practice, we are force to mine database in various situation, so data distribution is skewed. The dataset 1 amounts to 80MB and it is partitioned into 40MB, 20MB, 10MB and 10MB and allocated to over four nodes. Thus, each nodes has different size of dataset. We put least amount of data to node 4 while employing fast microprocessor in order to artificially generate skew. This is prepared for candidate migration experiments. The dataset 2 is 120MB, and is devided into 80MB,20MB,10MB and 10MB. Apparently experiment with dataset 2 has higher skew than that with dataset 1. This is used for transaction migration experiments. And in all of the experiments, the *skew* value was set to 0.2 [2]. Here we changed the size of data sets, just because the target data set could be different for each data mining applications. Thus uneven data distribution naturally happens in such situations.

	Node 1	Node 2	Node 3	Node 4
Proc.	P.Pro	P.Pro	P.Pro	P.II
Clock	200MHz	200MHz	200MHz	333MHz
Disk	SCSI	SCSI	SCSI	IDE
DataSet1	40MB	20MB	10MB	10MB
DataSet2	80MB	20MB	10MB	10MB

Table 1: Execution environment

	DataSet1	DataSet2
Number of transactions	1000000	1500000
Avg. size of transactions	20	20
Number of items	5000	5000

Table 2: Datasets

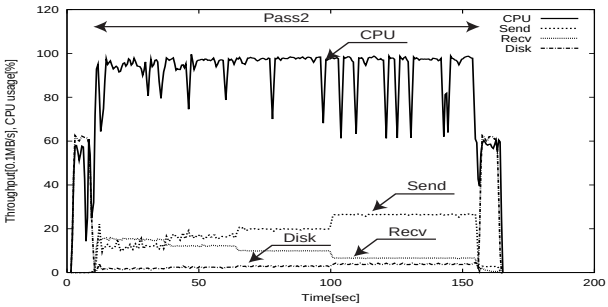
5.3 Performance evaluation results

Experiment with Dataset 1 for candidate migration

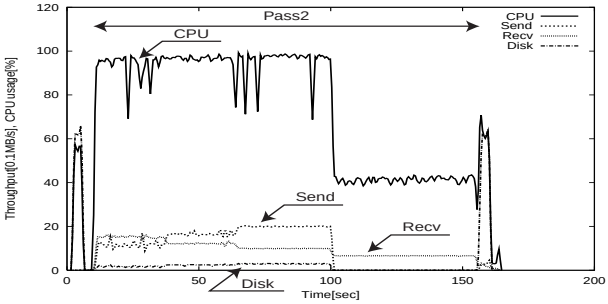
The numbers of candidate itemsets(C) and large itemsets(L) resulted from data mining of dataset 1 with 0.7% minimum support are shown in table 3. It is known that generally second pass produces the largest amount of candidate itemsets. The execution traces without any load migration are shown in Figure 2. The figure shows four different resource usage: CPU, disk, interconnection network (send/receive). Horizontal axis is elapsed time and vertical axis denotes utilization ratio for CPU and data transfer throughput for disk read and interconnection network. The network throughput is divided into two parts, send throughput and receive throughput.

Since we are activating four nodes, we could show four different traces. But we omit that for Node 3, since the space is limited. The figure shows that each node generates k-itemsets and sends them to destination nodes consecutively. In the first half of second pass Node 1 is too busy with RECV process receiving k-itemsets from other nodes, and could not even afford to read its own transaction data and perform SEND process. On the other hand, Node 4 with more powerful CPU and less data finishes reading its 10MB transaction data in first 40 seconds and spends the rest of time just waiting for incoming k-itemsets from other nodes. Node 2 has more work than Node 4 but

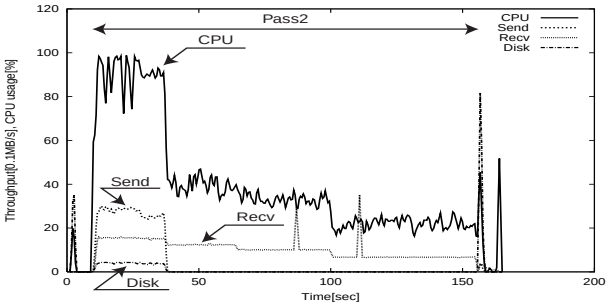
[2] In this case, result of some experiments, we set *skew* 0.2. But we think this parameter's tuning is one of the future works



(a)Node1

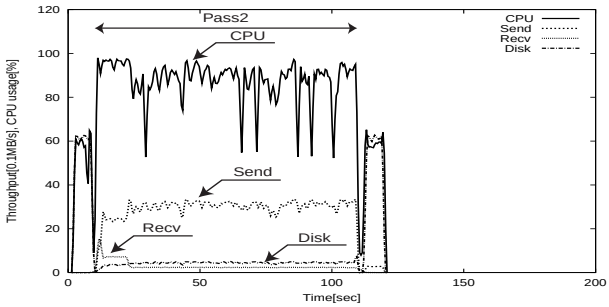


(b)Node2

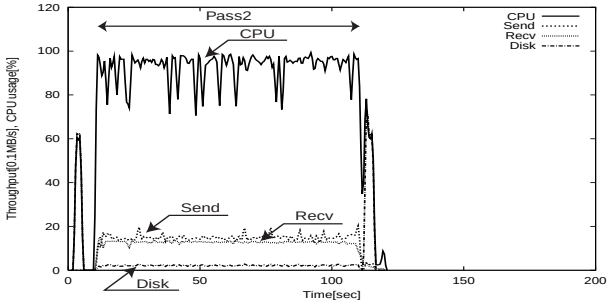


(b)Node4

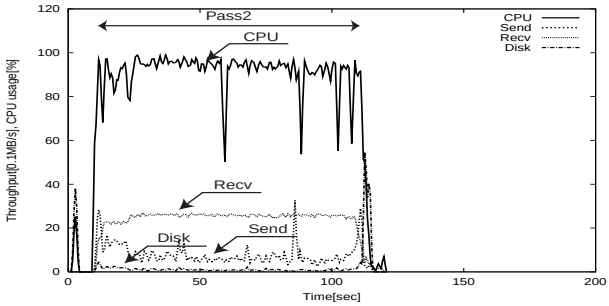
Figure 2: Execution trace without load balancing (DataSet1)



(a)Node1



(b)Node2



(b)Node4

Figure 3: Execution trace with Candidate Migration (DataSet1)

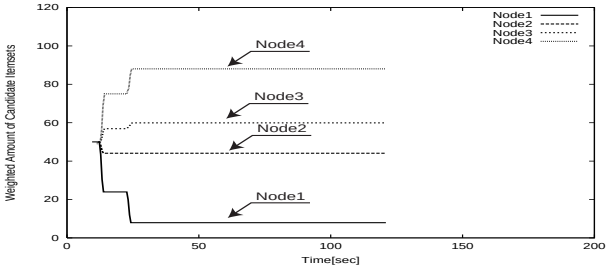


Figure 4: Migration trace for weighted amount of candidate itemsets(DataSet1)

	DataSet1		DataSet2	
	C	L	C	L
Pass 1	5000	989	5000	982
Pass 2	488566	54	481671	51
Pass 3	42	4	38	4
Pass 4	0	0	0	0

Table 3: Number of candidate itemset and large itemset for DataSet1 and DataSet2

it completes reading the transaction data much earlier than Node 1. The total execution time is 164.03 s.

When we apply Candidate Migration strategy, candidate itemsets are reallocated as soon as skew is detected. The traces are shown in Figure 3. Every node completes its task at almost the same time indicating the skew is eliminated and workload is evenly distributed. The processing time is also greatly improved; data mining with Candidate Migration requires only 120.21 s.

Figure 4 shows the trace of weighted candidate itemsets of all the nodes. We can see that Node 1 and Node 2 migrate their candidate itemsets to Node 3 and Node 4. The amount of migrated itemsets gradually increases and finally converged to a certain value. Currently the load skew is checked every 10 seconds. But for first several tens of seconds we should not wait 10 seconds, such a long time. We had better wait longer time for the system to reach certain stable condition, where we can calculate α values in the expression (15) precisely. However, if we wait too long, the nodes with faster processor and small amount of transactions will process their workload very quickly. So even though some of the parameters are a bit imprecise, we should migrate workload to light nodes as early as possible. After that, we can gradually tune the workload by performing additional migration.

During support counting, as shown Figure 3, we can see that Node 4 receives much more k-itemsets than it sends out. Since probing hash table for k-itemsets received from the other nodes precedes that for k-itemsets from node's own transaction data, reading transactions from disk is suppressed. Thus by comparing Figure 2 and 3, we can conclude that Candidate Migration strategy succeeds to evenly distributes the work load among the nodes.

Experiment with Dataset 2 for both candidate migration and transaction migration

We did an experiment with more skewed environment using dataset 2. Result of data mining using dataset 2 and 0.7% minimum support is also shown in table 3. It produces most of candidate itemsets during second pass like dataset 1. Node 1 is becoming the bottleneck of the parallelization as shown in Figure 5. Since the space is limited, we omit the traces of Node 2 and 3, and shows the behavior of only Node 1 and 4. The total execution time is 287.09 s.

By introducing the Candidate Migration, performance can be improved. The burden of workload is dispersed from Node 1 since the candidate itemsets are reallocated to other nodes, which is shown in Figure 6. The processing time is reduced to 198.36 s. However since the load is extremely concentrated at Node 1, as figure 6 shows, Candidate Migration alone can not get rid of that skew completely. Node 4 finishes reading out the transactions from disk at around 125 s. After

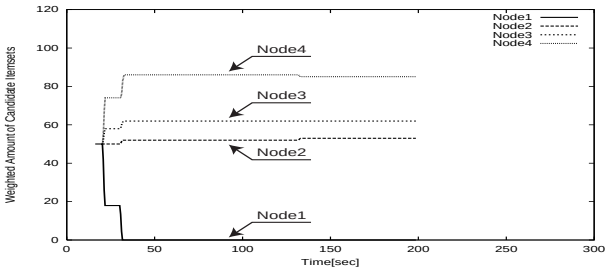


Figure 7: Migration trace for weighted amount of candidate itemsets(DataSet2)

that, it just receives the itemsets from other nodes. Thus the CPU usage goes down between 125 s. to 180 s. We can see that all candidate itemsets of Node 1 has been transferred to other nodes, as shown at figure 7. There remains no candidate itemsets at Node 1. Thus candidate migration cannot migrate workload any more. We need to introduce Transaction Migration.

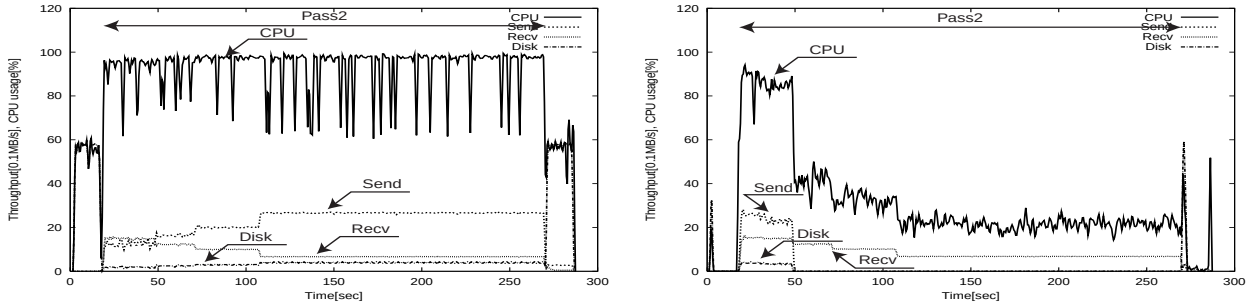
When we introduce transaction migration in addition to candidate migration, we can achive almost perfect load balancing as shown at Figure 8. Transaction migration works very effectively for highly skewed environment. Node 1 sends its transaction data and delegates the generation of k-itemsets to other nodes. The elimination of skew records processing time of 182.18 s. Transaction Migration can remove the workload skew which Candidate Migration is unable to handle.

Figure 9 shows the trace of amount of weighted candidate itemset and amount of migrated transactions for Node 1 and Node 4. No candidate itemsets is left at Node 1. Node 4 accepts candidate itemsets migrated from Node1. In addition to it, it receives the transaction given by Node 1. Node 1 sends out transactions to the other nodes and Node 4 receives some of the transactions from Node 1.

Experiment for scalability

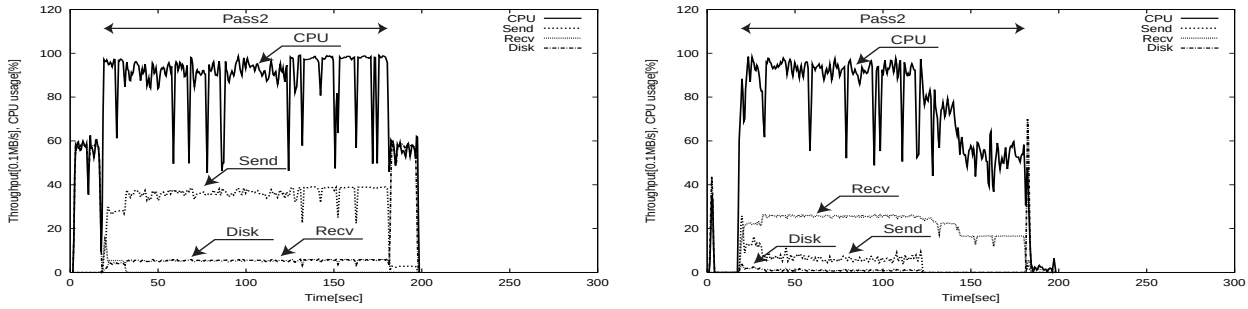
We have also examined the scalability of our strategies. We scaled up the system by multiplying the configuration we used so far. We used the configuration of a group of 4 nodes as multiplication unit and expanded the system from 4 nodes to 8, 12, 16, 24 and 32 nodes. We retained the composition of the 4 node system as described in table with dataset 1. Namely 8 node system is composed by just duplicating the original 4 node system. 12 node system is by replicating the original system three times and so on.

The results are shown in figure 10. The amount of transaction on each node does not change. So the total volume of transaction increases proportionally as the number of nodes increases. Execution time increases slightly as the number of nodes increases. As the number of nodes increases, the overhead time for synchronization becomes non-negligible. We think this is the



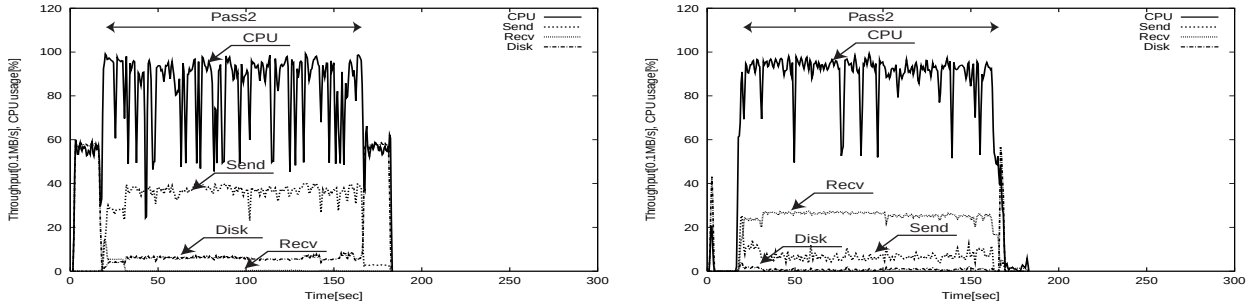
(a)Node1 (b)Node4

Figure 5: Execution trace without any load balancing(DataSet2)



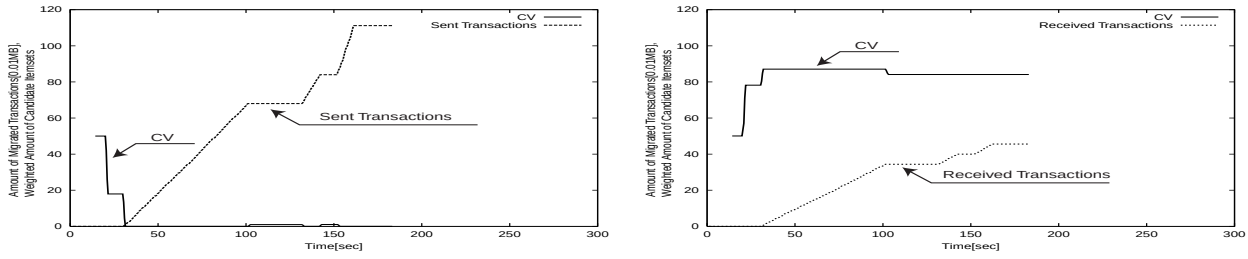
(a)Node1 (b)Node4

Figure 6: Execution trace with Candidate Migration(DataSet2)



(a)Node1 (b)Node4

Figure 8:Execution trace with both Candidate Migration and Transaction Migration(DataSet2)



(a)Node1 (b)Node4

Figure 9:Migration trace of weighted candidate itemsets and transactions(DataSet2)

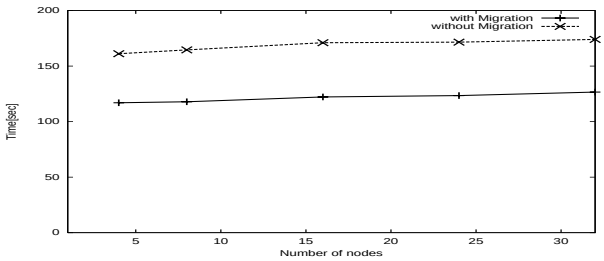


Figure 10: Scale-up results(DataSet1)

reason of slight performance degradation. We will describe this problem as a future extension at the conclusion. We are currently planing to introduce more nodes with 450MHz Pentium II and Xeon and perform larger scale experiment.

Experiment for the use of unused machines

So far we assumed that transaction database is partitioned over the nodes. Each node has some portion of transactions. We could use PC's which do not have transactions for data mining. In real situations, recent organizations have a lot of PC's and some of them are not used from time to time. We could make use of such idle PC's, in addition to the PC's which are originally assigned for data mining. This experiment tries to show how our scheme works in such environments.

Initially transactions are stored over two nodes. Then we add four idle nodes. Here in order to simplify the problem, we use same kinds of PC's. From the machine hardware type point of view, this might be homogeneous environment. But we store transactions on the disks of the two nodes but no transactions on the disks of other nodes, which can be regarded as heterogeneous. The number of transactions is 1 million. Figure 11 shows the experimental results. Figure 11(a) shows the execution trace of the experiment on only initial nodes. When we add the idle nodes, we can not exploit the resources of idel nodes effectively as shown figure 11(b). As you can see from the figure 11(c), by migrating the workload from the initial two nodes to the idle nodes, we can reduce the execution time and exploit CPU and memory resources of idle nodes effectively. And figure 11(d) shows the scale-up result, when initially 4 million transactions are stored over eight nodes, then we add idle nodes from one to sixteen. As shown figure 11(d), by migrating the workload from the initial nodes to the idle nodes, we can reduce the execution time significantly.

6 Conclusion

In this paper, we proposed dynamic load balancing strategies for parallel association rule mining on heterogeneous PC cluster system. Due to the short development period of recent PCs, it is inevitable that the PC cluster system becomes heterogeneous. Different types of CPUs are used from PC to PC. Different kinds of disks are also employed. In order to utilize all the system resources as fully as possible, we have to make the program adaptive to its runtime environment.

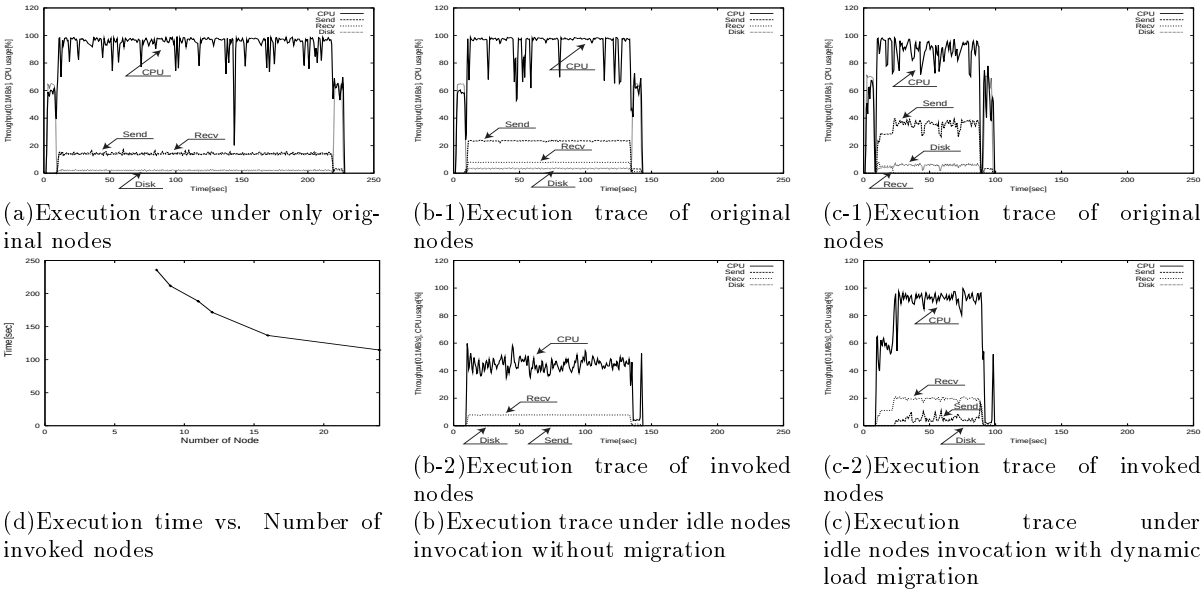
Compilation approach has its limitation, since the available resource might be different run by run. In addition, recent softwares have a lot of knobs, that is, tuning parameters, which makes system maintenance so difficult. This problem is pointed out also in Asilomar Report [16]. Thus the system had better adapt itself to the runtime environment autonomously.

In our proposed scheme, the parameters such as performance ratios are not necessary. At run time, the program derives several necessary coefficients by itself. Thus a programmer/compiler does not have to care about them. If we plan to use the unused system resources, say at night, the availability is unforseeable. In some case, other program might enter the system and start to run. So even during the execution, the available power might change. Our approach is designed to work even under such environment.

We adopted HPA(Hash Partitioned Apriori) algorithm for underlying parallel association rule mining. This partitions the candidate itemsets over the nodes, while ordinary methods just copy candidate itemsets all over the nodes. HPA can improve the memory efficiency significantly. We proposed two kinds of dynamic load balancing strategies for parallel association rule mining, Candidate Migration and Transaction Migration. We showed that by changing the allocation of candidate itemsets among the nodes, we can control the workload of PCs. The amount of candidates to be migrated can be derived analytically. If the skew is very high, Candidate Migration is not sufficient to balance the workload. In such case, we introduced yet another strategy named Transaction Migration. Since generation of candidate itemset from transaction is time consuming, heavy nodes send transactions to light nodes to whom itemset generation is delegated. This incurs extra data transmission but is effective to remove the workload skew.

In order to clearly show the effectiveness of our approach, we set up rather simple 4 node cluster with two kinds of PCs and varied the size of dataset for each PC. We demonstrated the feasibility of our approach showing the execution trace. By examining the trace, we confirmed that the proposed scheme effectively works to remove workload inbalance. Candidate Migration works under medium skew environment. If the skew is high and candidate migration can not sufficiently help, the system automatically invokes the Transaction Migration. In addition, we also showed the scalability experiments. We increased the size of the system from 4 nodes to 32 nodes. We found sufficient scalability can be archived

Currently, our algorithm estimates the itemset fre-



(a)Execution trace under only original nodes

(b-1)Execution trace of original nodes

(c-1)Execution trace of original nodes

(b-2)Execution trace of invoked nodes

(c-2)Execution trace of invoked nodes

(d)Execution time vs. Number of invoked nodes

(b)Execution trace under idle nodes invocation without migration

(c)Execution trace under idle nodes invocation with dynamic load migration

Figure 11: Experiment for the use of unused machines

quency using the support values of the previous pass. Based on this estimated itemset frequency, we derive the relative workload of candidate itemsets and determine the migration plan. Apparently this estimation can not be precise. During execution, we could modify the estimated value by using the runtime statistics. We are now examining the effect of estimation error and also implementing the extended version using runtime learning. Another extension is on the synchronization. In current implementation, the Candidate Migration is performed using barrier synchronization. That is, on migration all the nodes stop and migrate candidate simultaneously. Once migration completes, the system restarts again. If the size of the system becomes large, obviously it incurs a lot of overhead. So asynchronous migration should be employed, which we are going to investigate. And we also plan on implementing our ideas in generalized association rules[12], and sequential patterns. Still remains lots of interesting extensions, such as elimination of coordination nodes. Fully distributed algorithm is more challenging. So far we have focused on heterogeniety of hardware but same approach should work among PC's with different OS's. The number of nodes in our experiments is not necessarily large. The experimental results on 100 node environments is being undertaken, which will be reported in the future paper.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. "Fast Algorithms for Mining Association Rules". In *Proc. of VLDB* , pp. 487–499, Sep. 1994.

[2] Rakesh Agrawal and John C. Shafer. "Parallel Mining of Associaton Rules". In *IEEE TKDE*, Vol. 8, No. 6, pp. 962–969, Dec. 1996.

[3] *http://cesdis.gsfc.nasa.gov/beowulf/beowulf.html*

[4] D.W. Cheung, J. Han, V.T. Ng, A.W. Fu, and Y. Fu. "A Fast Distributed Algorithms for Mining Association Rules." In *Proc. of PDIS*, pp. 31–42, Dec. 1996.

[5] Hasanat M. Dewan, Mauricio A. Hernandez, Kui W. Mok, Salvatore J.Stolfo "Predictive Dynamic Load Balancing of Parallel Hash-Joins Over Heterogeneous Processors in the Presence of Data Skew." In *Proc. of PDIS*, pp. 40–49, 1994.

[6] D. DeWitt and J. Gray "Parallel Database Systems: The Future of High Performance Database Systems." In *Communications of the ACM*, Vol. 35, No. 6, pp. 85–98, Jun. 1992.

[7] E.-H.Han and G.Karypis and Vipin Kumar "Scalable Parallel Data Mining for Association Rules." In *Proc. of SIGMOD*, pp. 277–288, May. 1997

[8] Masaru Kitsuregawa, Takayuki Tamura, Masato Oguchi "Parallel Database Processing/Data Mining on Large Scale ATM Connected PC Cluster." In *Euro-PDS*, pp. 313–320, Jun. 1997

[9] J.S.Park, M.-S.Chen, P.S.Yu "Efficient Parallel Algorithms for Mining Association Rules" In *Proc. of CIKM*, pp. 31–36, Nov. 1995

[10] S.Parthasarathy and M.J.Zaki and W.Li "Memory Placement Techniques for Parallel Association Mining." In *Proc. of KDD*, pp. 304–308, Aug. 1998

[11] T. Shintani and M. Kitsuregawa "Hash Based Parallel Algorithms for Mining Association Rules". In *Proc. of PDIS*, pp. 19–30, Dec. 1996.

[12] T. Shintani, M. Kitsuregawa "Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy." In *Proc. of SIGMOD*, pp. 25–36, 1998.

[13] Takayuki Tamura, Masato Oguchi, Masaru Kitsuregawa "Parallel Database Processing on a 100 Node PC Cluster: Cases for Decision Support Query Processing and Data Mining." In *Super Computing 97::High Performance Networking and Computing*, 1997

[14] Yongqiao Xiao and David W. Cheung "Effect of Data Skewness in Parallel Data Mining of Association Rules ". In *Proc. of PAKDD*, pp. 48–60, Apr. 1998.

[15] M.J.Zaki, S.Parthasarathy, M.Ogihara and W.Li "New Parallel Algorithms for Fast Discovery of Association Rules". Data Mining and Knowledge Discovery, Dec. 1997.

[16] P.Bernstein, M.Brodie, S.Ceri, et.al "The Asilomar Report". Sep. 1998.