Hash Based Parallel Algorithms for Mining Association Rules

Takahiko SHINTANI Masaru KITSUREGAWA

The University of Tokyo, Institute of Industrial Science
3rd Dept., 7-22-1, Roppongi, Minato, Tokyo 106, Japan
Email : {shintani, kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract

In this paper, we propose four parallel algorithms (NPA, SPA, HPA and HPA-ELD) for mining association rules on shared-nothing parallel machines to improve its performance.

In NPA, candidate itemsets are just copied amongst all the processors, which can lead to memory overflow for large transaction databases. The remaining three algorithms partition the candidate itemsets over the processors. If it is partitioned simply (SPA), transaction data has to be broadcast to all processors. HPA partitions the candidate itemsets using a hash function to eliminate broadcasting, which also reduces the comparison workload significantly. HPA-ELD fully utilizes the available memory space by detecting the extremely large itemsets and copying them, which is also very effective at flattering the load over the processors.

We implemented these algorithms in a shared-nothing environment. Performance evaluations show that the best algorithm, HPA-ELD, attains good linearity on speedup ratio and is effective for handling skew.

1 Introduction

Recently, "Database Mining" has begun to attract strong attention. Because of the progress of bar-code technology, point-of-sales systems in retail company become to generate large amount of transaction data, but such data being archived and not being used efficiently. The advance of microprocessor and secondary storage technologies allows us to analyze this vast amount of transaction log data to extract interesting customer behaviors. Database mining is the method of efficient discovery of useful information such as rules and previously unknown patterns existing between data items embedded in large databases, which allows more effective utilization of existing data.

One of the most important problems in database mining is mining association rules within a database [1], so called the " basket data analysis" problem. Basket data type typically consist of a transaction identifier and the bought items par-transaction. By analyzing transaction data, we can extract the association rule such as "90% of the customers who buy both A and B also buy C".

Several algorithms have been proposed to solve the above problem[1][2][3][4][5][6][7]. However most of these are sequential algorithms. Finding association rules requires scanning the transaction database repeatedly. In order to improve the quality of the rule, we have to handle very large amounts of transaction data, which requires incredibly long computation time. In general, it is difficult for a single processor to provide reasonable response time. In [7], we examined the feasibility of parallelization of association rule mining[1]. In [6], a parallel algorithm called PDM, for mining association rules was proposed. PDM copies the candidate itemsets among all the processors. As we will explain later, in the second pass of the Apriori algorithm, introduced by R.Agrawal and R.Srikant[2], the candidate itemset becomes too large to fit in the local memory of a single processor. Thus it requires reading the transaction dataset repeatedly from disk, which results in significant performance degradation.

In this paper, we propose four different parallel algorithms (NPA, SPA, HPA and HPA-ELD) for mining association rules based on the Apriori algorithm. In NPA (Non Partitioned Apriori), the candidate itemsets are just copied among all the processors. PDM mentioned above corresponds to NPA. The remaining three algorithms partition the candidate itemsets over the processors. Thus exploiting the aggregate memory of the system effectively. If it is partitioned simply (SPA : Simply Partitioned Apriori), transaction data has to be broadcast to all the processors. HPA (Hash Partitioned Apriori) partitions the candidate itemsets using a hash function as in the hash join, which eliminates transaction data broadcasting and can reduce the comparison workload significantly. In case the size of candidate itemset is smaller than

[1]The paper was presented at a local workshop in Japan.

the available system memory, HPA does not use the remaining free space. However HPA-ELD (HPA with Extremely Large itemset Duplication) does utilize the memory by copying some of the itemsets. The itemsets are sorted based on their frequency of appearance. HPA-ELD chooses the most frequently occurring itemsets and copies them over the processors so that all the memory space is used, which contributes to further reduce the communication among the processor. HPA-ELD, an extension of HPA, treats the frequently occurring itemsets in a special way, which can reduce the influence of the transaction data skew.

The implementation on a shared-nothing 64-node parallel computer, the Fujitsu AP1000DDV, shows that the best algorithm, HPA-ELD, attains satisfactory linearity on speedup and is also effective at skew handling.

This paper is organized as follows. In next section, we describe the problem of mining association rules. In section 3, we propose four parallel algorithms. Performance evaluations and detail cost analysis are given in section 4. Section 5 concludes the paper.

2 Mining Association Rules

First we introduce some basic concepts of association rules, using the formalism presented in [1]. Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let $\mathcal{D} = \{t_1, t_2, \ldots, t_n\}$ be a set of transactions, where each transaction t is a sets of items such that $t \subseteq \mathcal{I}$. A transaction has an associated unique identifier called TID. We say each transaction *contains* a set of items X if $X \subseteq \mathcal{I}$. The itemset X has *support* s in the transaction set $\mathcal{D}$ if $s\%$ of transactions in $\mathcal{D}$ contain X, here we denote $s = support(X)$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X,\ Y \subset \mathcal{I}$, and $X \cap Y = \emptyset$. Each rule has two measures of value, *support* and *confidence*. The *support* of the rule $X \Rightarrow Y$ is $support(X \cup Y)$. The *confidence* c of the rule $X \Rightarrow Y$ in the transaction set $\mathcal{D}$ means $c\%$ of transactions in $\mathcal{D}$ that contain X also contain Y, which is 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 itemsets that have support above the user-specified minimum support. These itemset are called the *large itemsets*.

2. For each large itemset, derive all rules that have more than user-specified minimum confidence as follows: for a large itemset X and any Y $(Y \subset X)$, if $support(X)/support(X - Y) \geq minimum\_confidence$, then the rule $X - Y \Rightarrow Y$ is derived.

For example, let $T_1 = \{1,3,4\}$, $T_2 = \{1,2\}$, $T_3 = \{2,4\}$, $T_4 = \{1,2,3,5\}$, $T_5 = \{1,3,5\}$ be the transaction database. Let $minimum\_support$ and $minimum\_confidence$ be 60% and 70% respectively. Then, the first step generates the large itemsets $\{1\},\{2\},\{3\},\{1,3\}$. In the second step, an association rule $1 \Rightarrow 3$ ($support = 60\%, confidence = 75\%$) and $3 \Rightarrow 1$ ($support = 60\%, confidence = 100\%$) is derived.

After finding all large itemsets, association rules are derived in a straightforward manner. This second subproblem is not a big issue. However because of the large scale of transaction data sets used in database mining, the first subproblem is a nontrivial problem. Much of the research to date has focused on the first subproblem.

Here we briefly explain the Apriori algorithm for finding all large itemsets, proposed in [2], since the parallel algorithms to be proposed by us in section 3 are based on this algorithm. Figure 1 gives an overview of the algorithm, using the notation given in Table 1.

k-itemset	An itemset having k items.
L_k	Set of large k-itemsets, whose support is larger than user-specified minimum support.
C_k	Set of candidate k-itemsets, which is potentially large itemset

Table 1: Notation

In the first pass (pass 1), support_count for each item is counted 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 the items which satisfy the minimum support are picked out. These items are called large 1-itemset (L_1). Here k-itemset is defines a set of k items. The second pass (pass 2), the 2-itemsets are generated using the large 1-itemset which is called the candidate 2-itemsets (C_2). Then the support_count of the candidate 2-itemsets is counted by scanning the transaction database. Here support_count of the itemset means the number of transactions which contain the itemset. At the end of scan-

```
$L_1$ := large 1-itemsets
$k$ :=2
while ($L_{k-1} \neq \emptyset$) do
  $C_k$ := The candidates of size $k$ generated from $L_{k-1}$
  forall transactions $t \in \mathcal{D}$
    Increment the support_count of all candidates in
    $C_k$ that are contained in $t$
  $L_k$ := All candidates in $C_k$ which satisfy minimum
        support
  $k$ := $k$+1
end
Answer := $\bigcup_k L_k$
```

Figure 1: Apriori algorithm

ning the transaction data, the large 2-itemsets (L_2) which satisfy minimum support are determined. The following denotes the k-th iteration, pass k.

1. Generate candidate itemset:
   The candidate k-itemsets (C_k) are generated using large ($k-1$)-itemsets (L_{k-1}) which were determined in the previous pass (see Section 2.1).

2. Count support :
   The support_count for the candidate k-itemsets are counted 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 (L_k) which satisfy the minimum support are determined.

4. The procedure terminates when the large itemset becomes empty. Otherwise $k := k+1$ and goto "1".

2.1 Apriori Candidate Generation

The procedure for generating candidate k-itemsets using $(k-1)$-itemsets is as follows: Given a large $(k-1)$-itemset, we want to generate a superset of the set of all large k-itemsets. Candidate generation occurs in two steps. First, in the join step, join large $(k-1)$-itemset with $(k-1)$-itemset. Next, in the prune step, delete 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.

3 Parallel Algorithms

In this section, we describe four parallel algorithms (NPA, SPA, HPA and HPA-ELD) for the first subproblem, which we call count support processing hereafter, finding all large itemsets for shared-nothing parallel machines.

3.1 Algorithm Design

In the sequential algorithm, the count support processing requires the largest computation time, where the transaction database is scanned repeatedly and a large number of candidate itemsets are examined. We designed a parallel algorithm for count support processing.

If each processor can hold all of the candidate itemsets, parallelization is straightforward [2]. However for large scale transaction data sets, this assumption does not hold. Figure 2 shows the number of candidate itemsets and the large itemsets in each pass. These statistics were taken from the real point-of-sales data. In figure 2, the vertical axis is a log scale. The candi-

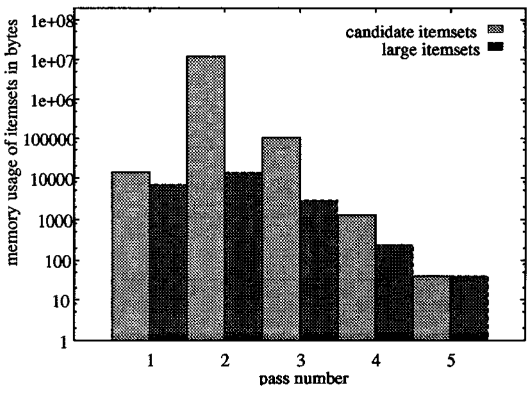

Figure 2: real point-of-sales data

date itemset of pass 2 is too large to fit within the local memory of a single processor. In NPA, the candidate itemsets are just copied amongst all the processors. In the case where all of the candidate itemsets do not fit within the local memory of a single processor, the candidate itemsets are partitioned into fragments, each of which fits in the memory size of a processor. Support count processing requires repetitive scanning transaction database. The remaining three algorithms, SPA, HPA and HPA-ELD, partition the candidate itemsets over the memory space of all the processors. Thus SPA, HPA and HPA-ELD can exploit the total system's memory effectively as the number of processors increases. For simplicity, we assume that the size of the candidate itemsets is larger than the size of local memory of single processor but is smaller than the sum of the memory of all the processors. It is easy

[2]We will later introduce an algorithm named NPA, where the reason why the parallelization is so easy will be clarified.

to extend this algorithm to handle candidate itemsets whose size exceeds the sum of all the processors memories.

3.2 Non Partitioned Apriori : NPA

In NPA, the candidate itemsets are copied over all the processors, each processor can work independently and the final statistics are gathered into a coordinator processor where minimum support conditions are examined. Figure 3 gives the behavior of pass k of the p-th processor in NPA, using the notation given in Table 2.

$\mathcal{L}_k$	Set of all the large k-itemsets.
$\mathcal{C}_k$	Set of all the candidate k-itemsets.
$\lvert\mathcal{C}_k\rvert$	The size of $\mathcal{C}_k$ in bytes.
M	The size of main memory in bytes.
$\mathcal{D}^p$	Transactions stored in the local disk of the p-th processor
C_k^d $(d = 1, \ldots, \lceil \frac{\lvert\mathcal{C}_k\rvert}{M} \rceil)$ $(\mathcal{C}_k = \bigcup_{d=1}^{\lceil \frac{\lvert\mathcal{C}_k\rvert}{M} \rceil} C_k^d)$	Sets of fragment of candidate k-itemsets. Each fragment fits in the local memory of a processor.
$\lvert C_k^d\rvert$	The size of C_k^d in bytes.
L_k^d	Sets of large k-itemsets derived from C_k^d.

Table 2: Notation

```
$\mathcal{C}_1$ := All items
$\{C_1^d\}$ := Partition $\mathcal{C}_1$ into fragments each of which
            fits in a processor's local memory
for($d = 1$; $d \le \lceil|\mathcal{C}_1|/M\rceil$; $d++$) do
  forall $t \in \mathcal{D}^p$ do
    Increment the support_count of all candidates in
      $C_1^d$ that are contained in $t$
  end
end
Send the support_count of $C_1^d$ to the coordinator
  /* Coordinator determine $L_1^d$ which satisfy
     user-specified minimum support in $C_1^d$ and
     broadcast $L_1^d$ to all processors            */
Receive $L_1^d$ from the coordinator
end
$\mathcal{L}_1 := \bigcup_d L_1^d$
$k := 2$
while ($L_{k-1} \neq \emptyset$) do
  $\mathcal{C}_k$ := The candidates of size $k$ generated from $L_{k-1}$
  $\{C_k^d\}_{(d=1,\ldots,\lceil|\mathcal{C}_k|/M\rceil)}$
   := Partition $\mathcal{C}_k$ into fragments each of which fits
      in a processor's local memory
  for($d = 1$; $d \le \lceil|\mathcal{C}_k|/M\rceil$; $d++$) do
    forall $t \in \mathcal{D}^p$ do
      Increment the support_count of all candidates in
        $C_k^d$ that are contained in $t$
    end
    Send the support_count of $C_k^d$ for to the coordinator
      /* Coordinator determine $L_k^d$ which satisfy
         user-specified minimum support in $C_k^d$ and
         broadcast $L_1^d$ to all processors            */
    Receive $L_k^d$ from the coordinator
  end
  $\mathcal{L}_k := \bigcup_d L_k^d$
  $k := k + 1$
end
```

Figure 3: NPA algorithm

Each processor works as follows:

1. Generate the candidate itemsets:
   Each processor generates the candidate k-itemsets using the large $(k-1)$-itemsets, and insert it into the hash table.

2. Scan the transaction database and count the support_count value:
   Each processor reads the transaction database from its local disk, generates k-itemsets from the transaction and searches the hash table. If a hit occurs, increment its support_count value.

3. Determine the large itemsets:
   After reading all the transaction data, all processor's support_count are gathered into the coordinator and checked to determine whether the minimum support condition is satisfied or not.

4. If large k-itemset is empty, the algorithm terminates. Otherwise $k := k + 1$ and the coordinator broadcasts large k-itemsets to all the processors and goto "1".

If the size of all the candidate itemsets exceeds the local memory of a single processor, the candidate itemsets are partitioned into fragments, each of which can fits within the processor's local memory and the above process is repeated for each fragment. Figure 3, beginning at the while loop, shows the method by which each of the candidate itemsets are divided into fragments with each fragment being processed sequentially.

Although this algorithm is simple and no transaction data are exchanged among processors in the second phase, the disk I/O cost becomes very large, since this algorithm reads the transaction database repeatedly if the candidate itemsets are too large to fit within the processor's local memory.

3.3 Simply Partitioned Apriori : SPA

In NPA, the candidate itemsets are not partitioned but just copied among the processors. However the candidate itemsets usually becomes too large to fit within the local memory of single processor, which generally occurs during the second pass ($k = 2$). SPA partitions the candidate itemsets equally over the memory space of all the processors. Thus it can exploit the aggregate memory of the systems, while memory efficiency is very low in copy based NPA.

Since the candidate itemsets are partitioned among the processors, each processor has to be broadcast its own transaction data to all the processors at second phase, while no such broadcast is required in NPA. Figure 4 gives the behavior of pass k by the p-th processor in SPA, using the notation in Table 3. Here we assume the size of candidate itemset is smaller than the size of sum of all the processor's memory. Extension of the algorithm to handle much larger candidate itemset is easy. We can divide the candidate itemsets into fragments like in NPA.

$\mathcal{L}_k$	Set of all the large k-itemsets.
$\mathcal{C}_k$	Set of all the candidate k-itemsets.
$\mathcal{D}^p$	Transactions stored in the local disk of the p-th processor
C_k^p ($c_k = \bigcup_{p=1}^{N} c_k^p$)	Sets of candidate k-itemsets assigned the p-th processor (N means the number of processors)
L_k^p	Sets of large k-itemsets derived from C_k^p

Table 3: Notation

Each processor works as follows:

1. Generate the candidate itemsets:
   Each processor generates the candidate k-itemsets using the large $(k-1)$-itemsets and inserts a part of the candidate itemsets into its own hash table. The candidate k-itemsets are assigned to processors in a round-robin manner [3].

2. Scan the transaction database and count the support_count value:

[3]The k-itemsets are assigned equally to all of the processors in a round-robin manner. By round-robin we mean that the candidates are assigned to the processors in a cyclical manner with the i-th candidate assigned to processor i mod n, where n is the number of processors in the system.

```
{$C_1^p$}:= All items assigned to the $p$-th processor
forall $t \in \mathcal{D}^p$ do
  Broadcast $t$ to all the other processors
  Receive the transaction sent from the other pro-
  cessors and increment the support_count of all
  candidates that are contained in received trans-
  action
end
{$L_1^p$}:= All the candidates in $C_1^p$ which satisfy
        user-specified minimum support
  /* Each processor can determine individ-
     ually whether assigned candidate $k$-
     itemset satisfy user-specified minimum */
     support or not
Send $L_1^p$ to the coordinator
  /* Coordinator make up $\mathcal{L}_1 := \bigcup_p L_1^p$ and
     broadcast it to all the other processors */
Receive $\mathcal{L}_1$ from the coordinator
while ($\mathcal{L}_{k-1} \neq \emptyset$) do
  {$C_k^p$}:= The candidates of size $k$, assigned to the
          $p$-th processor, which is generated from
          $L_{k-1}$
  forall $t \in \mathcal{D}^p$ do
    Broadcast $t$ to all the processors
    Receive the transaction sent from the other pro-
    cessors and increment the support_count of all
    candidates that are contained in the received
    transaction
  end
  {$L_k^p$}:= All the candidates in $C_k^p$ which satisfy the
          user-specified minimum support
  Send $L_k^p$ to the coordinator
    /* Coordinator make up $\mathcal{L}_k := \bigcup_p L_k^p$ and
       broadcast it to all the processors        */
  Receive $\mathcal{L}_k$ from the coordinator
  $k := k + 1$
end
```

Figure 4: SPA algorithm

Each processor reads the transaction database from its local disk and also broadcasts it to all the other processors. For each transaction entry, when read from its own disk or received from another processors, the support_count is incremented in the same way as in NPA.

3. Determine the large itemsets:
   After reading all the transaction data, each processor can determine individually whether each candidate k-itemset satisfy user-specified minimum support or not. Each processor send L_k^p to the coordinator, where $\mathcal{L}_k := \bigcup_p L_k^p$ are derived.

4. If large k-itemset is empty, the algorithm terminates. Otherwise $k := k+1$ and the coordinator broadcasts large k-itemsets to all the processors and goto "1".

Although this algorithm is simple and easy to implement, the communication cost becomes very large, since this algorithm broadcasts all the transaction data at second phase.

3.4 Hash Partitioned Apriori : HPA

HPA partitions the candidate itemsets among the processors using the hash function like in the hash join, which eliminates broadcasting of all the transaction data and can reduce the comparison workload significantly. Figure 5 gives the behavior of pass k by the p-th processor in HPA, using the notation in Table 3.

Each processor works as follows:

1. Generate the candidate itemsets:
   Each processor generates the candidate k-itemset using the large $(k-1)$-itemsets, applies the hash function and determines the destination processor ID. If the ID is its own, insert it into the hash table. If not, it is discarded.

2. Scan the transaction database and count the support_count:
   Each processor reads the transaction database from its local disk. Generates k-itemsets from that transaction and applies the same hash function used in phase 1. Derives the destination processor ID and sends the k-itemset to it. For the itemsets received from the other processors and those locally generated whose ID equals the processor's own ID, search the hash table. If hit, increment its support_count value.

3. Determine the large itemset:
   Same as in SPA.

```
{$C_1^p$}:= All items assigned to the $p$-th processor
        based on hashed value
forall $t \in \mathcal{D}^p$ do
  forall items $x \in t$ do
    Determine the destination processor ID by apply-
    ing the same hash function which is used in item
    partitioning, and send that item to it. If it is its
    own ID, increment the support_count for the item.
    Receive the item from the other processors and in-
    crement the support_count for that item
  end
end
{$L_1^p$}:= All the candidates in $C_1^p$ with minimum sup-
        port
  /* Each processor can determine individu-
     ally whether assigned candidate $k$-itemset
     satisfy user-specified minimum support or
     not                                        */
Send $L_1^p$ to the coordinator
  /* Coordinator make up $\mathcal{L}_1 := \bigcup_p L_1^p$ and
     broadcast to all the processors            */
Receive $\mathcal{L}_1$ from the coordinator
while ($\mathcal{L}_{k-1} \neq \emptyset$) do
  {$C_k^p$}:= All the candidate $k$-itemsets, whose
          hashed value corresponding to the $p$-th
          processor
  forall $t \in \mathcal{D}^p$ do
    forall $k$-itemset $x \in t$ do
      Determine the destination processor ID by ap-
      plying the same hash function which is used in
      item partitioning, and send that $k$-itemset to it.
      If it is its own ID, increment the support_count
      for the itemset.
      Receive $k$-itemset from the other processors and
      increment the support_count for that itemset
    end
  end
  {$L_k^p$}:= All the candidates in $C_k^p$ with minimum
          support
  Send $L_k^p$ to the coordinator
    /* Coordinator make up $\mathcal{L}_k := \bigcup_p L_k^p$ and
       broadcast to all the processors          */
  Receive $\mathcal{L}_k$ from the coordinator
  $k := k+1$
end
```

Figure 5: HPA algorithm

4. If large k-itemset is empty, the algorithm terminates. Otherwise $k := k + 1$ and the coordinator broadcasts large k-itemsets to all the processors and goto "1".

3.5 HPA with Extremely Large Itemset Duplication : HPA-ELD

In case the size of candidate itemset is smaller than the available system memory, HPA does not use the remaining free space. However HPA-ELD does utilize the memory by copying some of the itemsets. The itemsets are sorted based on their frequency of appearance. HPA-ELD chooses the most frequently occurring itemsets and copies them over the processors so that all the memory space is used, which contributes to further reduce the communication among the processor. In HPA, it is generally difficult to achieve a flat workload distribution. If the transaction data is highly skewed, that is, some of the itemsets appear very frequently in the transaction data, the processor which has such itemsets will receive a much larger amount of data than the others. This might become a system bottleneck. In real situations, the skew of items is easily discovered. In retail applications certain items such as milk and eggs appear more frequently than others. HPA-ELD can handle this problem effectively since it treats the frequently occurring itemset entries in a special way.

HPA-ELD copies such frequently occurring itemsets among the processors and counts the support_count locally like in NPA. In the first phase, when the processors generate the candidate k-itemset using the large $(k-1)$-itemsets, if the sum of the support values for each large itemset exceeds the given threshold, it is inserted in all the processor's hash table. The remaining candidate itemsets are partitioned as in HPA. The threshold is determined so that all of the available memory can be fully utilized using sort. After reading all the transaction data, all processor's support_count are gathered and checked whether it satisfies the minimum support condition or not. Since most of the algorithm steps are equal to HPA, we omit a detailed description of HPA-ELD.

4 Performance Evaluation

Figure 6 shows the architecture of Fujitsu AP1000DDV system, on which we have measured the performance of the proposed parallel algorithms for mining association rules, NPA, SPA, HPA and HPA-ELD. AP1000DDV employs a shared-nothing architecture. A 64 processor system was used, where each processor, called cell, is a 25MHz SPARC with 16MB local memory and a 1GB local disk drive. Each processor is connected to three independent networks (T-net, B-net and S-net). The communication between processors is done via a torus mesh network called the T-net, and broadcast communication is done via the B-net. In addition, a special network for barrier synchronization, called the S-net is provided.

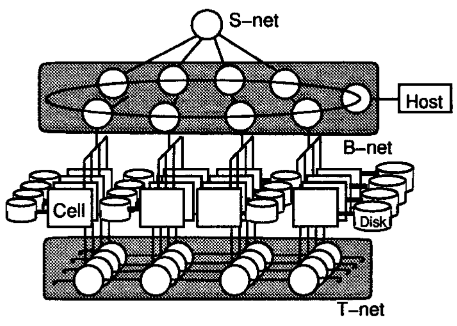

Figure 6: Organization of the AP1000DDV system

$\lvert\mathcal{D}\rvert$	the number of transactions
$\lvert t\rvert$	the average number of items in partransactions
$\lvert I\rvert$	the average number of items in maximal potentially large itemsets

Name	$\lvert t\rvert$	$\lvert I\rvert$	$\lvert\mathcal{D}\rvert$	Size
t10.I4	10	4	2048K	100MB
t15.I4	15	4	2048K	145MB
t20.I4	20	4	2048K	187MB

Table 4: Parameters of data sets

To evaluate the performance of the four algorithms, synthetic data emulating retail transactions is used, where the generation procedure is based on the method described in [2]. Table 4 shows the meaning of the various parameters and the characteristics of the data set used in the experiments.

4.1 Measurement of Execution Time

Figure 7 shows the execution time of the four proposed algorithms using three different data sets with varying minimum support values. $16(4 \times 4)$ processors are used in these experiments. Transaction data is evenly spread over the processor's local disks. In these experiments, each parallel algorithm is adopted only for pass 2, the remaining passes are performed using NPA, since the single processor's memory cannot hold the entire candidate itemsets only for pass 2 and if it fits NPA is most efficient.

HPA and HPA-ELD significantly outperforms SPA.

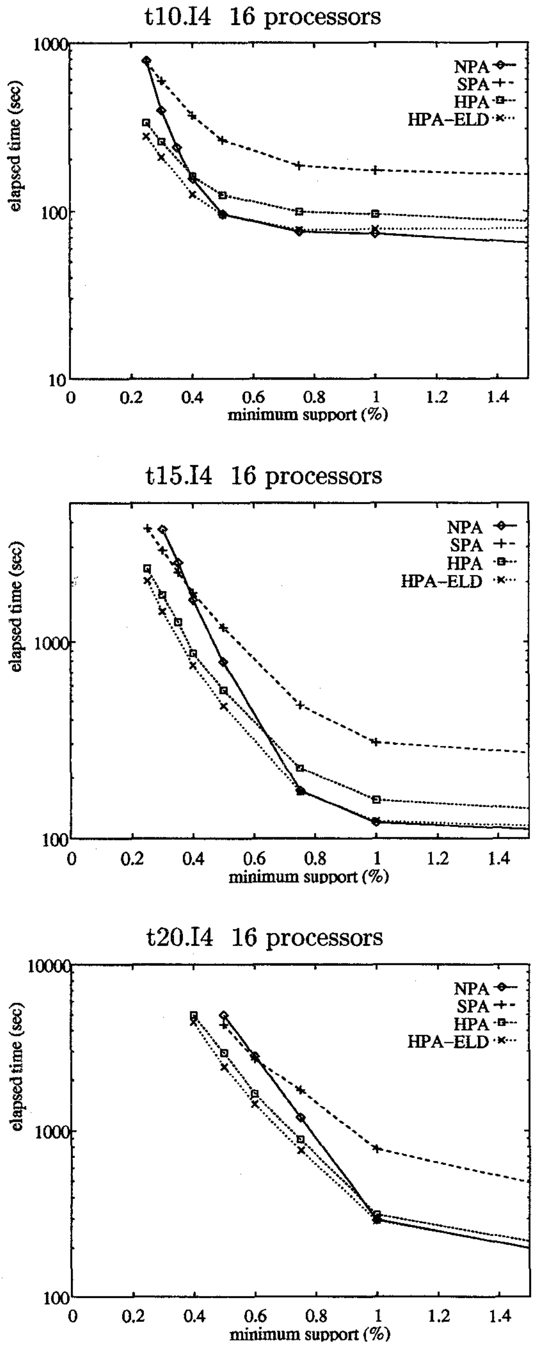

Figure 7: Execution time varying minimum support value

Since all transaction data is broadcast to all of the processors in SPA, its communication costs are much larger in SPA than in HPA and HPA-ELD where the data is not broadcasted but transfered to just one processor determined by a hash function. In addition SPA transmits the transaction data, while HPA and HPA-ELD transmit the itemsets, which further reduces the communication costs.

In NPA, the execution time increases sharply when the minimum support becomes small. Since the candidate itemsets becomes large for small minimum support, the single processor's memory cannot hold the entire candidate itemsets. NPA has to divide the candidate itemsets into fragments. Processors have to scan the transaction data repetitively for each fragment, which significantly increases the execution time.

4.2 Communication Cost Analysis

Here we analyze the communication costs of each algorithm. Since the size of the transaction data is usually much larger than that of the candidate itemset, we focus on the transaction data transfer. In NPA, the candidate itemsets are initially copied over all the processors, which incurs processor communication. In addition during the last phase of the processing, each processor sends the support count statistics to the coordinator where the minimum support condition is examined. This also incurs communications overhead. But here we ignore such overhead and concentrate on the transaction data transfer for SPA and HPA in second phase.

In SPA, each processor broadcasts all transaction data to all the other processors. The total amount of communication data of SPA at pass k can be expressed as follows.

$$\begin{aligned} M_k^{SPA} &= \sum_{p=1}^{N}\sum_{i=1}^{T_p} t_{ip} \times (N-1) \\ &\simeq |t| \times (N-1) \times |\mathcal{D}| \qquad (1) \end{aligned}$$

where

p	processor ID ($p = 1, 2, \ldots, N$)
N	the number of processors
t_{ip}	the number of items in i-th transaction of p-th processor
T_p	the number of p-th processor's transactions
$\lvert\mathcal{D}\rvert$	the number of all the transactions ($\lvert\mathcal{D}\rvert = \sum_p T_p$)

In HPA, the itemsets of the transaction are transmitted to the limited number of processors instead of broadcasting. The number of candidates is dependent on the data synthesized by the generator. The total amount of communication for HPA at pass k can be expressed as follows.

$$M_k^{HPA} = \sum_{p=1}^{N}\sum_{i=1}^{T_p} {}_{t_{ip}}C_k \times k \times \alpha_{ip}^k$$
$$\simeq {}_{|t|}C_k \times k \times |\alpha^k| \times |\mathcal{D}| \qquad (2)$$

One transaction potentially generate ${}_{t_{ip}}C_k$ candidates. However in practice most of them are filtered out, as is denoted by the parameter α_{ip}^k. Since α is usually small[4], $M_k^{SPA} \gg M_k^{HPA}$. Since it is difficult to derive α, we measured the amount of data received by each processor. Figure 8 shows the total amounts of received messages of SPA, HPA and HPA-ELD where t15.I4 transaction data was used with 0.4% minimum support. As you can see in Figure 8, the amount of messages received of HPA is much smaller then that of SPA. In HPA-ELD, the amount of messages received is further reduced, since a part of the candidate itemset is handled separately and the itemsets which correspond to them are not transmitted but just locally processed.

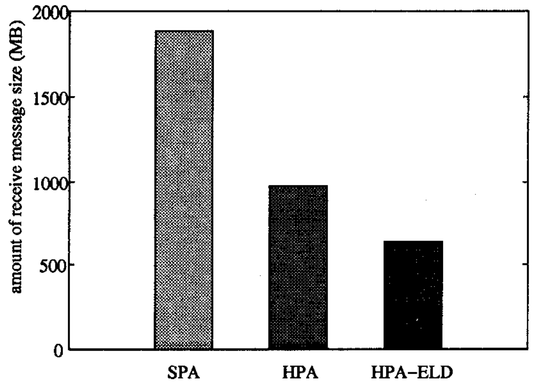

Figure 8: the amount of messages received (pass 2)

4.3 Search Cost Analysis

In the second phase, the hash table which consists of the candidate itemsets are probed for each transaction itemset.

[4]If the number of processors is very small and the number of items in transaction is large, then M_k^{HPA} could be larger than M_k^{SPA} With reasonable number of processors, this does not happen as you can see in Figure 8. We are currently doing experiments on mining association rules with item's classification hierarchy, where combination of items becomes much larger than the ordinary mining association rules.

When α_k increases, M_k^{HPA} tends to increase as well. We will report on this case in a future paper.

In NPA, the number of probes at pass k can be expressed as follows.

$$S_k^{NPA} = \left\lceil \frac{CAN}{M} \right\rceil \sum_{p=1}^{N}\sum_{i=1}^{T_p} {}_{t_{ip}}C_k \times \alpha_{ip}^k$$
$$\simeq {}_{|t|}C_k \times |\alpha^k| \times |\mathcal{D}| \times \left\lceil \frac{CAN}{M} \right\rceil \qquad (3)$$

where

CAN	the amount of the candidate itemset in bytes
M	the size of main memory of a single processor in bytes

In NPA, if the candidate itemsets are too large to fit in a single processor's memory, the candidate itemsets are divided and the supports are counted by scanning the transaction database repeatedly.

In SPA, every processor must process all the transaction data. The number of searches at pass k can be expressed as follows.

$$S_k^{SPA} = N \times \sum_{p=1}^{N}\sum_{i=1}^{T_p} {}_{t_{ip}}C_k \times \alpha_{ip}^k$$
$$\simeq {}_{|t|}C_k \times |\alpha^k| \times |\mathcal{D}| \times N \qquad (4)$$

In HPA and HPA-ELD, the number of searches at pass k can be expressed as follows.

$$S_k^{HPA} = \sum_{p=1}^{N}\sum_{i=1}^{T_p} {}_{t_{ip}}C_k \times \alpha_{ip}^k$$
$$\simeq {}_{|t|}C_k \times |\alpha^k| \times |\mathcal{D}| \qquad (5)$$

The search cost of HPA and HPA-ELD is always smaller than SPA. It is apparent that $S_k^{HPA} < S_k^{SPA}$. Not only the communication cost but also search cost also can be reduced significantly by employing hash based algorithms, which is quite similar to the way in which the hash join algorithm works much better than nested loop algorithms. In NPA, the search cost depends on the size of the candidate itemsets. If the candidate itemset becomes too large, S_k^{NPA} could be larger than S_k^{SPA}. But if it fits, $S_k^{NPA} \simeq S_k^{HPA} < S_k^{SPA}$, that is, the search cost is much smaller than SPA and almost equal to HPA. Figure 9 shows the search cost of the three algorithms for each pass, where the t15.I4 data set is used under 16 processors with the minimum support 0.4%. In the experimental results we have so far shown, all passes except pass 2 adopts NPA algorithm. We applied different algorithms only for pass 2, which is computationally heaviest part of the total processing. However, here in order to focus on the search cost of individual algorithm more clearly, each algorithm is applied for all passes. The cost of

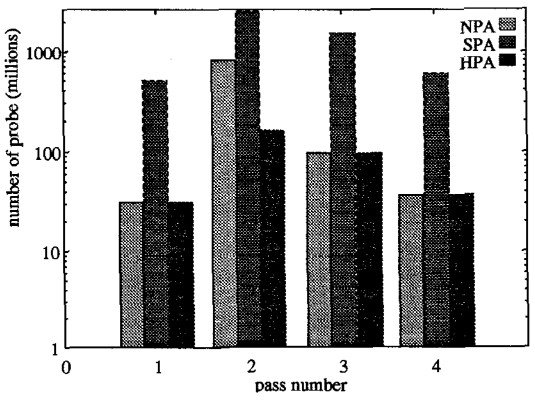

Figure 9: the search cost of SPA, NPA and HPA

NPA changes drastically for pass 2. The search cost of NPA is highly dependent on the size of available main memory. If memory is insufficient, NPA's performance deteriorates significantly due to the cost increase at pass 2. In Figure 9, the search cost of NPA is less than SPA. However as we explained before, it incurred a lot of additional I/O cost. Therefore the total execution time of NPA is much longer than that of SPA.

4.4 Comparison of HPA and HPA-ELD

In this section, the performance comparison between HPA and HPA-ELD is described. In HPA-ELD, we treat the most frequently appearing itemsets separately. In order to determine which itemset we should pick up, we use the statistics accumulated during pass 1. As the number of pass increases, the size of the candidate itemsets decreases. Thus we focused on pass 2. The number of the candidate itemsets to be separated is adjusted so that sum of non-duplicated itemsets and duplicated itemsets would just fit in the available memory.

Figure 10 shows the execution time of HPA and HPA-ELD for t15.I4 varying the minimum support value on a 16 processors system. HPA-ELD is always faster than HPA. The smaller the minimum support, the larger the ratio of the difference between the execution times of the two algorithms becomes. As the minimum support value decreases, the number of candidate itemsets and the count of support increases. The candidate itemsets which are frequently found cause large amounts of communication. The performance of HPA is degraded by this high communications traffic.

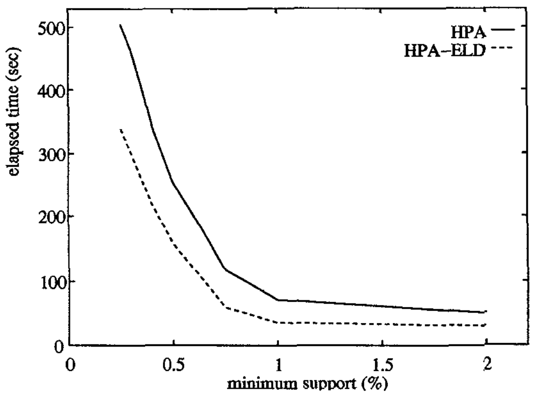

Figure 10: Execution time of HPA and HPA-ELD at pass 2

Figure 11 shows the number of probes in each processor for HPA and HPA-ELD for t15.I4 using a 16 processor system for pass 2. We picked up an example which is highly skewed. Horizontal axis denotes processor ID. In HPA, the distribution of the number

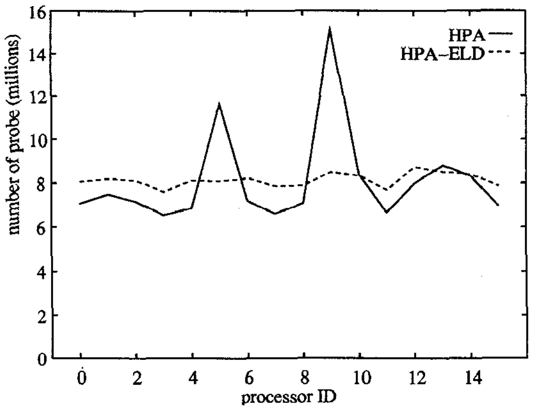

Figure 11: The number of search of HPA and HPA-ELD at pass 2

of probes is not flat. Since each candidate itemset is allocated to just one processor, the large amount of messages concentrate at a certain processor which has many candidate itemsets occurring frequently.

In HPA-ELD, the number of probes is comparatively flat. HPA-ELD handle certain candidate itemsets separately, thus reducing the influence of the data skew. However, as you can see in Figure 11, there still remain the deviation of the load amongst processors. If we parallelize the mining over more than 64 processors, we have to introduces more sophisticated load balancing mechanism, which requires further investigation.

4.5 Speedup

Figure 12 shows the speedup ratio for pass 2 varying the number of processors used, 16, 32, 48 and 64, where the curve is normalized with the 16 processor execution time. The minimum support value was set to 0.4%.

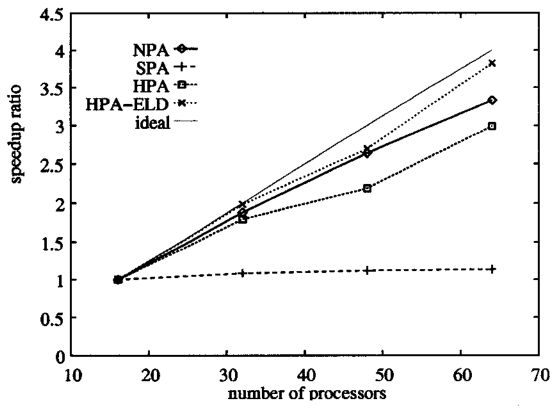

Figure 12: Speedup curve

NPA, HPA and HPA-ELD attain much higher linearity than SPA. HPA-ELD, an extension of HPA for extremely large itemset decomposition further increases the linearity.

HPA-ELD attains satisfactory speed up ratio. This algorithm just focuses on the item distribution of the transaction file and picks up the extremely frequently occurring items. Transferring such items could result in network hot spots. HPA-ELD tries not to send such items but to process them locally. Such a small modification to the original HPA algorithm could improve the linearity substantially.

4.6 Effect of increasing transaction database size (Sizeup)

Figure 13 shows the effect of increasing transaction database size as the number of transactions is increased from 256,000 to 2 million transactions. We used the data set t15.I4. The behavior of the results does not change with increased database size. The minimum support value was set to 0.4%. The number of processors is kept at 16. As shown, each of the parallel algorithms attains linearity.

5 Summary and related work

In this paper, we proposed four parallel algorithms for mining association rules. A summary of the four

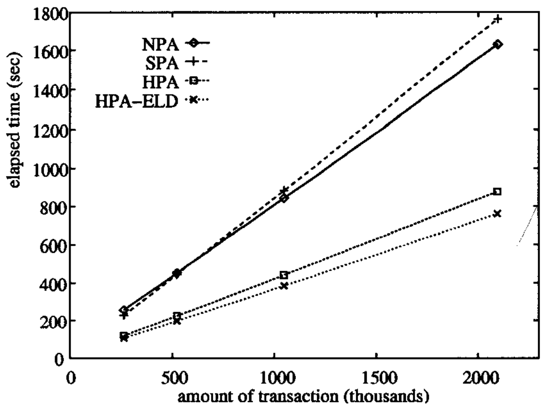

Figure 13: Sizeup curve

algorithms is shown in Table 5. In NPA, the candidate itemsets are just copied amongst all the processors. Each processor works on the entire candidate itemsets. NPA requires no data transfer when the supports are counted. However in the case where the entire candidate itemsets do not fit within the memory of a single processor, the candidate itemsets are divided and the supports are counted by scanning the transaction database repeatedly. Thus Disk I/O cost of NPA is high. PDM, proposed in [6] is the same as NPA, which copies the candidate itemsets among all the processors. Disk I/O for PDM should be also high.

The remaining three algorithms, SPA, HPA and HPA-ELD, partition the candidate itemsets over the memory space of all the processors. Because it better exploits the total system's memory, disk I/O cost is low. SPA arbitrarily partitions the candidate itemsets equally among the processors. Since each processor broadcasts its local transaction data to all other processors, the communication cost is high. HPA and HPA-ELD partition the candidate itemsets using a hash function, which eliminates the need for transaction data broadcasting and can reduce the comparison workload significantly. HPA-ELD detects frequently occurring itemsets and handles them separately, which can reduce the influence of the workload skew.

6 Conclusions

Since mining association rules requires several scans of the transaction file, its computational requirements are too large for a single processor to have a reasonable response time. This motivates our research.

In this paper, we proposed four different parallel algorithms for mining association rules on a shared-nothing parallel machine, and examined their viabil-

	NPA	SPA	HPA	HPA-ELD
Candidate itemset	copy	partition		partition (partially copy)
I/O cost	high	low		
Communication cost	-	high	low	
Skew handling	×			○

Table 5: characteristics of algorithms

ity through implementation on a 64 node parallel machine, the Fujitsu AP1000DDV.

If a single processor can hold all the candidate itemsets, parallelization is straightforward. It is just sufficient to partition the transaction over the processors and for each processor to process the allocated transaction data in parallel. We named this algorithm NPA. However when we try to do large scale data mining against a very large transaction file, the candidate itemsets become too large to fit within the main memory of a single processor. In addition to the size of a transaction file, a small minimum support also increases the size of the candidate itemsets. As we decrease the minimum support, computation time grows rapidly, but in many cases we can discover more interesting association rules.

SPA, HPA and HPA-ELD not only partition the transaction file but partition the candidate itemsets among all the processors. We implemented these algorithms on a shard-nothing parallel machine. Performance evaluations show that the best algorithm, HPA-ELD, attains good linearity on speedup by fully utilizing all the available memory space, which is also effective for skew handling. At present, we are doing the parallelization of mining generalized association rules described in [9], which includes the taxonomy (is-a hierarchy). Each item belongs to its own class hierarchy. In such mining, associations between the higher class and the lower class are also examined. Thus the candidate itemset space becomes much larger and its computation time also takes even longer than the naive single level association mining. Parallel processing is essential for such heavy mining processing.

Acknowledgments

This research is partially supported as a priority research program by ministry of education. We would like to thank the Fujitsu Parallel Computing Research Center for allowing us to use their AP1000DDV systems.

References

[1] R.Agrawal, T.Imielinski, and A.Swami: "Mining Association Rules between Sets of Items in Large Databases", In *Proc. of the 1993 ACM SIGMOD International Conference on Management of Data*, pp207-216, May 1993.

[2] R.Agrawal, and R.Srikant: "Fast Algorithms for Mining Association Rules", In *Proc. of the 20th International Conference on Very Large Data Bases*, pp.487-499, September 1994.

[3] J.S.Park, M.-S.Chen, and P.S.Yu: "An Effective Hash-Based Algorithm for Mining Association Rules", In *Proc. of the 1995 ACM SIGMOD International Conference on the Management of Data*, SIGMOD Record Vol.24, pp.175-186, June 1995.

[4] H.Mannila, H.Toivonen, and A.I.Verkamo: "Efficient Algorithms for Discovering Association Rules", In *KDD-94:AAAI Workshop on Knowledge Discovery in Databases*, pp.181-192, July 1994.

[5] A.Savasere, E.Omiecinski, and S.Navathe: "An Effective Algorithm for Mining Association Rules in Large Databases", In *Proc. of the 21th International Conference on Very Large Data Bases*, pp.432-444, September 1995.

[6] J.S.Park, M.-S.Chen, and P.S.Yu: "Efficient Parallel Data Mining for Association Rules", In *Proc. of the 4th International Conference on Information and Knowledge Management*, pp.31-36, November 1995.

[7] T.Shintani, and M.Kitsuregawa: "Consideration on Parallelization of Database Mining", In *Institute of Electronics, Information and Communication Engineering Japan (SIG-CPSY95-88)*, Technical Report, Vol.95, No.47, pp.57-62, December 1995.

[8] T.Shimizu, T.Horie, and H.Ishihata: "Performance Evaluation of the AP1000 -Effects of message handling, broadcast, and barrier synchronization on benchmark performance-", In *SWoPP'92(92-ARC-95) Information Processing Society of Japan*, Vol.92, No.64, 1992.

[9] R.Srikant and R.Agrawal: "Mining Generalized Association Rules", In *Proc. of the 21th International Conference on Very Large Data Bases*, pp.407-419, September 1995.