Effect of Data Distribution in Parallel Mining of Associations

David W. Cheung† Yongqiao Xiao†

† Department of Computer Science and Information Systems,

The University of Hong Kong, Hong Kong. Email: {dcheung}@csis.hku.hk.

Abstract

Association rule mining is an important new problem in data mining. It has crucial applications in decision support and marketing strategy. We proposed an efficient parallel algorithm for mining association rules on a distributed share-nothing parallel system. Its efficiency is attributed to the incorporation of two powerful candidate set pruning techniques. The two techniques, distributed and global prunings, are sensitive to two data distribution characteristics: data skewness and workload balance. The prunings are very effective when both the skewness and balance are high. We have implemented FPM on an IBM SP2 parallel system. The performance studies show that FPM outperforms CD consistently, which is a parallel version of the representative Apriori algorithm [2]. Also, the results have validated our observation on the effectiveness of the two pruning techniques with respect to the data distribution characteristics. Furthermore, it shows that FPM has nice scalability and parallelism, which can be tuned for different business applications.

Keywords: Association Rules, Data Mining, Data Skewness, Workload Balance, Parallel Mining, Paralle Computing.

1 Introduction

Association rule discovery has attracted a lot of attention from the research and business communities [1, 2, 4]. An association rule is a rule which implies certain association relationships among a set of objects (such as "occur together" or "one implies the other") in a database. The intuitive meaning of an association $X \Rightarrow Y$, where X and Y are set of items, is that transactions of the database which contain X tend to contain Y. A classical example is that 98% of customers that purchase tires and automobile accessories in a department store also have automotive services carried out. This example is a typical association in a basket database which sounds like common sense knowledge; however, there could be a lot of associations among the data which may not be able to deduce from common knowledge. Therefore, efficient automated technique to discover this type of rules is a very important area of research in data mining [2, 7, 8, 15, 17]. Applications of association rule mining range from decision support to product marketing and consumer behavior prediction.

Previous studies examined efficient mining of association rules from many different angles. An influential association rule mining algorithm, Apriori [2], has been developed for rule mining in large transaction databases. The scope of the study has also been extended to efficient mining of sequential patterns [20], generalized association rules

[19], multiple-level association rules [9], quantitative association rules [21], constrainted association rules [14] etc. Although these studies are on sequential data mining techniques, algorithms for parallel mining of association rules have also been proposed [3, 16, 22, 23].

The development of parallel systems for mining of association rules has its unique importance — databases or data warehouses [18] have been used more often to store a huge amount of data; data mining in such databases require substantial processing power, and parallel system is a possible solution. This observation motivates us to study efficient parallel algorithms for mining association rules in large databases. In this work, we study the problem on parallel system with distributed share-nothing memory such as the IBM SP2 [11]. In this model, the database is partitioned and distributed across the local disks of the processors; and the processors communicate via a fast network.

It has been well studied that the major cost of mining association rules is the computation of the set of *large itemsets* (i.e., *frequently occurring sets of items*, see Section 2.1) in the database [1, 2]. An itemset (a set of items) is *large* if the percentage of transactions that containing all these items is greater than a given threshold. The most representative parallel algorithm for mining association rules is the CD algorithm (Count Distribution), which is designed for share-nothing parallel systems [3]. It extends directly the basic technique of Aprori to parallel system. Our proposed algorithm, FPM (Fast Parallel Mining), has the following distinct feature in comparison with CD : FPM has explored an important property between *locally large itemsets* (those that are large with respect to the partition of a processor) and *globally large itemsets* (those that are large with respect to the entire database) to develop two powerful pruning techniques, *distributed pruning* and *global pruning*, which can reduce the number of candidate sets at each individual processor. Since the number of candidate sets is a dominant parameter of the computation cost, with a substantially smaller candidate sets, FPM performs much better than CD.

Another contribution of this work is the discovery that the effectiveness of the two aforementioned pruning techniques, and hence the performance of the parallel mining, depends on the data distribution characteristics in the database partitioning. We have captured the distribution characteristics in two factors : *data skewness* and *workload balance*. These two factors are orthogonal properties. Intuitively, a partitioned database has high data skewness if most globally large itemsets are locally large only at a very few partitions. On the other hand, a partitioned database has a high workload balance if all the processors have similar number of locally large itemsets in their partitions. (More precise definitions of skewness and workload balance will be given in Sections 3 & 4.) We have defined metrics to measure data skewness and workload balance. We found out that both the distributed and global prunings have super performance in the best case of high data skewness and high workload balance. The combination of high balance with moderate skewness is the second best case. On the other hand, the high skewness, moderate balance combination only provide moderate improvement over CD, while the combination of low skewness and low balance is the worst case in which only marginal improvement can be found.

We have implemented FPM on an IBM SP2 parallel machine with 32 processors. Extensive performance studies have been carried out. The results confirm our observation on the relationship between pruning effectiveness and data distribution.

The rest of this paper is organized as follows. Section 2 overviews the parallel mining of association rules. The techniques of distributed and global prunings, together with the FPM algorithm are described in Section 3. In the same section, we have also investigated the relationship between the effectiveness of the prunings and the two data distribution characteristics of data skewness and workload balance. In section 4, we define metrics to measure data skewness and workload balance of a data partition. Section 5 reports the result of an extensive performance study.

In Section 6, we discuss a few issues including possible extensions of FPM to enhance its scalability. Section 7 is the conclusion.

2 Parallel Mining of Association Rules

2.1 Sequential Algorithm for Mining Association Rules

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of *items*. Let D be a database of transactions, where each transaction T consists of a set of items such that $T \subseteq I$. Given an *itemset* $X \subseteq I$, a transaction T *contains* X if and only if $X \subseteq T$. An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$ [2]. The association rule $X \Rightarrow Y$ holds in D with *confidence* c if the probability of a transaction in D which contains X also contains Y is c. The association rule $X \Rightarrow Y$ has *support* s in D if the probability of a transaction in D contains both X and Y is s. The task of mining association rules is to find all the association rules whose support is larger than a *minimum support threshold* and whose confidence is larger than a *minimum confidence threshold*.

For an itemset X, its *support* is the percentage of transactions in D which contains X, and its *support count*, denoted by $X_{.sup}$, is the number of transactions in D containing X. An itemset X is *large* (or more precisely, *frequently occurring*) if its support is no less than the minimum support threshold. An itemset of size k is called a *k-itemset*. It has been shown that the problem of mining association rules can be reduced to two subproblems [1]: (1) *find all large itemsets for a given minimum support threshold*, and (2) *generate the association rules from the large itemsets found*. Since (1) dominates the overall cost of mining association rules, the research has been focused on developing efficient methods to solve the first subproblem [2].

An interesting serial algorithm, *Apriori* [2], has been proposed for computing large itemsets at mining association rules in a transaction database, which is outlined as follows [2].

> The large itemsets are computed through iterations. At each iteration, Apriori scans the database once and finds all the large itemsets of the same size. At the k-th iteration, Apriori creates the set of candidate sets $C_{(k)}$ by applying the candidate set generating function *Apriori_gen* on $L_{(k-1)}$, where $L_{(k-1)}$ is the set of all large $(k-1)$-itemsets found at the $(k-1)$-st iteration, and Apriori_gen generates only those k-itemset candidates whose every $(k-1)$-itemset subset is in $L_{(k-1)}$.

2.2 Count Distribution Algorithm for Parallel Mining

CD (Count Distribution) is a parallel version of Apriori. It is one of the earliest proposed and representative parallel algorithms for mining of association rules [3]. We describe here briefly its steps for comparison purpose. The database D is partitioned into $D_1, D_2, \cdots, D_n$ and distributed across n processors. The program fragment of CD at processor p_i, $1 \leq i \leq n$, for the k-th iteration is outlined in Figure 1. (For convenience, we use $X_{.sup(i)}$ to represent the local support count of an itemset X in partition D_i.) In step 1, every processor computes the same candidate set C_k by applying the aprior_gen function on L_{k-1}, which is the set of large itemsets found at the $(k-1)$-th iteration. In step 2, local support counts (support in D_i) of candidates in C_k are found. In steps 3 & 4, local support counts are exchanged with all other processors to get global support counts (support in D) and globally large itemsets (large with respect to D) L_k are computed independently by each processor. CD repeats steps 1 – 4 until no more candidate is found. We have implemented CD on an IBM SP2 using the MPI (Message Passing Interface) [13].

1) $C_k = \text{apriori\_gen}(L_{k-1})$;
2) scan partition D_i to find the local support count $X_{.sup(i)}$ for all $X \in C_k$;
3) exchange $\{X_{.sup(i)} \mid X \in C_k\}$ with all other processors to get global support counts $X_{.sup}$, for all $X \in C_k$;
4) $L_k = \{X \in C_k \mid X_{.sup} \geq minsup \times |D|\}$

Figure 1: Count Distribution Algorithm

3 Pruning Techniques and the FPM Algorithm

3.1 Distributed Pruning

It is important to observe some interesting properties related to large itemsets in a parallel environments since such properties may substantially reduce the number of candidate sets. (The preliminary form of the results in this section have been developed in [6] and extended here.) First, there is an important relationship between large itemsets and the processors in the database: *every globally large itemsets must be locally large at some processor(s).* If an itemset X is *both globally large and locally large* at a processor p_i, X is called gl-large at processor p_i. The set of gl-large itemsets at a processor will form a basis for the processor to generate its own candidate sets.

Second, a gl-large itemset at a processor has the following monotonic subset relationship property: *if an itemset is gl-large at a processor p_i, all of its subsets are also gl-large at p_i.* Combining these two properties, we have the following results.

Lemma 1 *If an itemset X is globally large, there exists a processor p_i, $(1 \leq i \leq n)$, such that X and all its subsets are gl-large at processor p_i.*[1]

We use GL_i to denote the set of gl-large itemsets at processor p_i, and $GL_{i(k)}$ to denote the set of gl-large k-itemsets at processor p_i. It follows from Lemma 1 that if $X \in L_{(k)}$, then there exists a processor p_i, such that all its size-$(k-1)$ subsets are gl-large at processor p_i, i.e., they belong to $GL_{i(k-1)}$.

In a straightforward adaptation of Apriori, the set of candidate sets at the k-th iteration, denoted by $CA_{(k)}$, which stands for size-k candidate sets from Apriori, would be generated by applying the Apriori_gen function on $L_{(k-1)}$. That is,

$$CA_{(k)} = \text{Apriori\_gen}(L_{(k-1)}).$$

At each processor p_i, let $CG_{i(k)}$ be the set of candidates sets generated by applying Apriori_gen on $GL_{i(k-1)}$, i.e.,

$$CG_{i(k)} = \text{Apriori\_gen}(GL_{i(k-1)}),$$

where CG stands for candidate sets generated from gl-large itemsets. Hence $CG_{i(k)}$ is generated from $GL_{i(k-1)}$. Since $GL_{i(k-1)} \subseteq L_{(k-1)}$, $CG_{i(k)}$ is a subset of $CA_{(k)}$. In the following, we use $CG_{(k)}$ to denote the set $\cup_{i=1}^{n} CG_{i(k)}$.

Theorem 1 *For every $k > 1$, the set of all large k-itemsets $L_{(k)}$ is a subset of $CG_{(k)} = \bigcup_{i=1}^{n} CG_{i(k)}$, where $CG_{i(k)} = \text{Apriori\_gen}(GL_{i(k-1)})$.*

[1]This result is stronger than that in [17] — the result there states that a globally large itemset is locally large in some partition; while Lemma 1 states that all its subsets must be locally large together at the same partition.

Proof. Let $X \in L_{(k)}$. It follows from Lemma 1 that there exists a processor p_i, $(1 \le i \le n)$, such that all the size-$(k-1)$ subsets of X are gl-large at processor p_i. Hence $X \in CG_{i(k)}$. Therefore,

$$L_{(k)} \subseteq CG_{(k)} = \bigcup_{i=1}^{n} CG_{i(k)} = \bigcup_{i=1}^{n} \text{Apriori\_gen}(GL_{i(k-1)}).$$

□

Theorem 1 indicates that $CG_{(k)}$, which is a subset of $CA_{(k)}$ and could be much smaller than $CA_{(k)}$, can be taken as the set of candidate sets for the size-k large itemsets. In effect, the set of candidates in $CA_{(k)}$ has been pruned down to those in $CG_{(k)}$ — we called this technique *distributed pruning*. This result forms a basis for the reduction of the set of candidate sets in the algorithm FPM. First the set of candidate sets $CG_{i(k)}$ can be generated locally at each processor p_i at the k-th iteration. After the exchange of support counts, the gl-large itemsets $GL_{i(k)}$ in $CG_{i(k)}$ can be found at the end of that iteration. Based on $GL_{i(k)}$, the candidate sets at processor p_i for the $(k+1)$-st iteration can then be generated according to Theorem 1. According to our performance studies, the number of candidate sets generated by distributed pruning can be substantially reduced to about $10-25\%$ of that generated in CD.

Example 1 illustrates the effectiveness of the reduction of candidate sets using distributed pruning.

Example 1 Assuming there are 3 processors in a parallel system in which the database D has been partitioned into D_1, D_2 and D_3. Suppose the set of large 1-itemsets (computed at the first iteration) $L_{(1)} = \{A, B, C, D, E, F, G, H\}$, in which A, B, and C are locally large at processor p_1, B, C, and D are locally large at processor p_2, and E, F, G, and H are locally large at processor p_3. Therefore, $GL_{1(1)} = \{A, B, C\}$, $GL_{2(1)} = \{B, C, D\}$, and $GL_{3(1)} = \{E, F, G, H\}$.

Based on Theorem 1, the set of size-2 candidate sets at processor p_1 is $CG_{1(2)} = \text{Apriori\_gen}\ (GL_{1(1)}) = \{AB, BC, AC\}$. Similarly, $CG_{2(2)} = \{BC, CD, BD\}$, and $CG_{3(2)} = \{EF, EG, EH, FG, FH, GH\}$. Hence, the set of candidate sets for large 2-itemsets is $CG_{(2)} = CG_{1(2)} \cup CG_{2(2)} \cup CG_{3(2)}$, total 11 candidates.

However, if Apriori_gen is applied to $L_{(1)}$, the set of candidate sets $CA_{(2)} = \text{Apriori\_gen}(L_{(1)})$ would have 28 candidates. This shows that it is very effective to apply distributed pruning to reduce the candidate sets. □

3.2 Global Pruning

As a result of the count exchange, the local support counts $X_{.sup(i)}$, for all processor p_i, $(1 \le i \le n)$, are also available at every processor. With this information, another powerful pruning technique called *global pruning* can be developed. Let X be a candidate k-itemset. At each partition D_i, $X_{.sup(i)} \le Y_{.sup(i)}$, if $Y \subset X$. Therefore the local support count of X, $X_{.sup(i)}$, is bounded by the value $\min\{Y_{.sup(i)} \mid Y \subset X, \text{ and } |Y| = k-1\}$. Since the global support count of X, $X_{.sup}$, is the sum of its local support count at all the processors, the value

$$X_{.maxsup} = \sum_{i=1}^{n} X_{.maxsup(i)}, where\ X_{.maxsup(i)} = \min\{Y_{.sup(i)} \mid Y \subset X, \text{ and } |Y| = k-1\},$$

is an upper bound of $X_{.sup}$. If $X_{.maxsup} < \text{minsup} \times |D|$, then X can be pruned away. This technique is called *global pruning*. Note that global pruning requires no additional information except the local support counts resulted from count exchange in the previous iteration.

Table 1 gives an example to show that global pruning can pruning away candidates which cannot be pruned by distributed pruning. Suppose the global support count threshold is 15 and the local support count threshold at each

Items	A	B	C	D	E	F
local support at processor 1	13	33	1	2	2	1
local support at processor 2	3	3	12	34	1	4
local support at processor 3	2	3	2	1	12	33
global support	18	39	15	37	15	38
gl-large at processor 1	$\surd$	$\surd$	$\times$	$\times$	$\times$	$\times$
gl-large at processor 2	$\times$	$\times$	$\surd$	$\surd$	$\times$	$\times$
gl-large at processor 3	$\times$	$\times$	$\times$	$\times$	$\surd$	$\surd$

Table 1: High data skewness and high workload balance case

processor is 5. Distributed pruning cannot prune away CD, as C and D are both gl-large at processor 2. Whereas global pruning can prune away CD, as $CD_{.maxsup} = CD_{.maxsup(1)} + CD_{.maxsup(2)} + CD_{.maxsup(3)} = 1 + 12 + 1 = 14 < 15$. EF can also be pruned, because $EF_{.maxsup} = 1 + 1 + 12 = 13 < 15$. However, AB would survive global pruning. From this example, it is clear that global pruning is more effective than distributed pruning, i.e., what can pruned away by distributed pruning will be pruned away by global pruning. The three pruning techniques, the one in apriori_gen, the distributed and global prunings, have increasing pruning power, and the latter ones subsume the previous one.

3.3 Fast Parallel Mining Algorithm (FPM)

We present the FPM algorithm in this section. It is an enhancement of CD. The simple support counts exchange scheme in CD is retained in FPM. The main difference is the incorporation of both the distributed and global prunings in FPM to reduce the candidate set size.

The first iteration of FPM is the same as CD. Each processor scans its partition to find out local support counts of all size-1 itemsets and use one round of count exchange to compute the global support counts. At the end of the 1-st iteration, in addition to L_1, each processor also finds out the gl-large itemsets $GL_{1(i)}$, for $1 \leq i \leq n$.

For the k-th iteration of FPM, $k > 1$, the program fragment executed at processor i, $1 \leq i \leq n$, is described in Figure 2.

1) compute candidate sets $CG_k = \cup_{i=1}^{n}$ apriori_gen($GL_{k-1(i)}$); (distributed pruning)
2) prune candidates in CG_k by global pruning;
3) scan partition D_i to find the local support count $X_{.sup(i)}$ for all remaining candidates $X \in CG_k$;
4) exchange $\{X_{.sup(i)} \mid X \in CG_k\}$ with all other processors to get global support counts $X_{.sup}$, for all $X \in CG_k$;
5) compute $GL_{k(i)} = \{X \in CG_k \mid X_{.sup} \geq minsup \times |D|, X_{.sup(i)} \geq minsup \times |D_i|\}$, for all i, $1 \leq i \leq n$;
6) return $L_k = \cup_{i=1}^{n} GL_{k(i)}$.

Figure 2: The FPM Algorithm

Similar to CD, FPM is also implemented by collective communication operations of MPI on the SP2.

In order to compare the effects of distributed and global pruning, we have also implemented a variant FNG (FPM with no global pruning) of FPM. FNG does not perform the global pruning, i.e., it's procedure is the same as that in Figure 2, except step 2 is removed.

4 Data Skewness and Workload Balance

In a database partition, two data distribution characteristics, *data skewness* and *workload balance*, have orthogonal effects on prunings and hence performance of FPM.

Intuitively, the data skewness of a partitioned database is high if the supports of most large itemsets are clustered in a few partitions. It is low if the supports of most large itemsets are distributed evenly across the processors. In Table 1, it is clear that all the itemsets have high skewness.

For a partition with high skewness, even though the support of each large itemset is clustered at a few processors, the clusterings of different large itemsets may be distributed evenly across the processors or concentrated on a few of them. In the first case, the clusterings of the large itemsets are distributed evenly among the processors; hence, each processor would have similar number of locally large itemsets. We characterise this case as high workload balance. In the second case, the clusterings would be concentrated on a few processors; hence some processors would have much more locally large itemsets than the others. This is the low workload balance case. For example, the itemsets in Table 1 not only have high skewness, it also has a good workload balance; because A, B are locally large at processor 1, and C, D at processor 2, whereas E, F at processor 3.

It follows from our discussion of the pruning techniques that high data skewness would increase the chance of candidate sets pruning; however, it is not the only factor, workload is another critical factor. In the following, we will see that given a good data skewness, if the distribution of the clusterings amount the processors are not even, then the pruning effects would be reduced significantly, and, to aggravate the problem more, the work of computing the large itemsets would be concentrated on a few processors which is a very troublesome issue for parallel computation.

Example 2 As explained above, Table 1 is a case of high data skewness and high workload balance. The supports of each itemset are distributed mostly in one partition; hence, the skewness is high. On the other hand, every partition has the same number of locally large itemsets; therefore, the workload balance is also high. In this case, CD will generate $\binom{6}{2} = 15$ candidates in the second iteration. Whereas, the distributed pruning will generate only three candidates AB, CD and EF, which shows that the pruning has good effect for this distribution.

Items	A	B	C	D	E	F
local support at processor 1	13	33	12	34	2	1
local support at processor 2	1	3	1	2	1	4
local support at processor 3	2	1	2	1	12	33
global support	16	37	15	37	15	38
gl-large at processor 1	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	$\times$
gl-large at processor 2	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
gl-large at processor 3	$\times$	$\times$	$\times$	$\times$	$\surd$	$\surd$

Table 2: High data skewness and low workload balance case

Table 2 is an example of high data skewness but low workload balance. The thresholds are the same as that in Table 1, i.e., the global support threshold is 15 and the local support threshold at each processor is 5. The support count distribution of each item is the same as that in Table 1 except that items A, B, C and D are now locally large together at processor 1 instead of distributed between processors 1 and 2. In this lower workload balance case, distributed pruning will generate 7 size-2 candidates, namely AB, AC, AD, BC, BD, CD and EF, while CD will still have 15 candidates. Thus, the distributed pruning remains to be very effective, but not as good as that in the high workload balance case (Table 1).

Items	A	B	C	D	E	F
local support at processor 1	6	12	4	13	5	12
local support at processor 2	6	12	5	12	4	13
local support at processor 3	4	13	6	12	6	13
global support	16	37	15	37	15	38
gl-large at processor 1	$\surd$	$\surd$	$\times$	$\surd$	$\surd$	$\surd$
gl-large at processor 2	$\surd$	$\surd$	$\surd$	$\surd$	$\times$	$\surd$
gl-large at processor 3	$\times$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

Table 3: Low data skewness and high workload balance case

Table 3 is an example of low data skewness and high workload balance. The support counts of the items A, B, C, D, E and F are almost equally distributed over the 3 processors. Hence, the data skewness is low. On the other hand, the workload balance is high, because the number of locally large itemsets in each processor is almost the same. In this case, both CD and distributed pruning generate the same 15 candidate sets; hence, if we restrict pruning to the distributed pruning, then it has no advantage over CD in this case. However, global pruning can prune away the candidates AC, AE and CE. In other words, FPM still has a 20% of improvement over CD in this pathological case of low skewness and high balance. □

Following Example 2, it is observed that global pruning is more effective than distributed pruning and can perform significant candidates reduction even in the moderate data skewness or low workload balance cases. As a note, low skewness and low balance cannot occur together. Also, according to our analysis, distributed pruning can prune away almost $\frac{n-1}{n}$ (n is the number of partitions) of all the size-2 candidates generated by CD in the high data skewness and high workload balance case.

In summary, distributed pruning is very effective when a database is partitioned with high skewness and high balance. On the other hand, in the worst cases of high skewness with low balance or high balance with low skewness, the effect of distributed pruning is degraded to the level in CD, however, global pruning may still perform better than CD. To strengthen our studies, we investigated the problem of defining metrics to measure skewness and balance.

4.1 Data Skewness Metric

We have developed a skewness metric based on the well established notion of entropy [5]. Given a random variable X, it's entropy is a measurement on how even or uneven its probability distribution is over its values. If a database is partitioned over n processors, the value $p_X(i) = \frac{X_{.sup(i)}}{X_{.sup}}$ can be regarded as the probability of occurrence of an itemset X in partition D_i, $(1 \leq i \leq n)$. The entropy $H(X) = -\sum_{i=1}^{n}(p_X(i) \times log(p_X(i)))$ is a measurement of the distribution of the local supports of X over the partitions. For example, if X is skewed completely into a single partition D_k, $(1 \leq k \leq n)$, i.e., it only occurs in D_k, then $p_X(k) = 1$ and $p_X(i) = 0$, $\forall i \neq k$. The value of $H(X) = 0$

is the minimal in this case. On the other hand, if X is evenly distributed among all the partitions, then $p_X(i) = \frac{1}{n}$, $1 \le i \le n$, and the value of $H(X) = log(n)$ is the maximal in this case. Therefore the following metric can be used to measure the skewness of a data partition.

	Itemset	A	B	C	D	E	F	workload W_i
Table 1	local count at processor 1	13	33	1	2	2	1	0.329
	local count at processor 2	1	3	12	34	1	4	0.348
high	local count at processor 3	2	1	2	1	12	33	0.322
data skewness	$S(X)$	0.452	0.633	0.429	0.697	0.429	0.586	
high	$TS(D)$	0.494						
workload balance	$TB(D)$	0.999						
Table 2	local count at processor 1	13	33	12	34	2	1	0.601
	local count at processor 2	1	3	1	2	1	4	0.076
high	local count at processor 3	2	1	2	1	12	33	0.323
data skewness	$S(X)$	0.452	0.633	0.429	0.697	0.429	0.586	
low	$TS(D)$	0.494						
workload balance	$TB(D)$	0.789						
Table 3	local count at processor 1	6	12	4	13	5	12	0.329
	local count at processor 2	6	12	5	12	4	13	0.329
low	local count at processor 3	4	13	6	12	6	13	0.342
data skewness	$S(X)$	0.015	0.001	0.012	0.001	0.012	0.001	
high	$TS(D)$	0.005						
workload balance	$TB(D)$	0.999						

Table 4: Data Skewness and Workload Balance

Definition 1 *Given a database partition D_i, $(1 \le i \le n)$, the skewness $S(X)$ of an itemset is defined by* $S(X) = \frac{H_{max} - H(X)}{H_{max}}$, *where* $H(X) = -\sum_{i=1}^{n}(p_X(i) \times log(p_X(i)))$ *and* $H_{max} = log(n)$.

The skewness $S(X)$ has the following properties:

- $S(X) = 0$, when all $p_X(i)$, $1 \le i \le n$, are equal. So the skewness is at its lowest value when X is distributed evenly in all partitions.
- $S(X) = 1$, if $\exists k \in$ [1, n] such that $p_X(k) = 1$, and $p_X(i) = 0$ for $\forall i \ne k, 1 \le i \le n$. So the skewness is at its highest value when X occurs only in one partition.
- $0 < S(X) < 1$, in all the other cases.

It follows from the property of entropy that $S(X)$ increases with respect to the skewness of X; hence, it is a suitable metric for the skewness of an individual itemset. Table 4 shows the skewness of the large itemsets in Tables 1, 2 and 3.

In addition to measuring the skewness of an itemset, we also need a metric to measure the skewness of the database partition. We define the skewness of a database partition as a weighted sum of the skewness of all the large itemsets. In other words, the skewness of a partition is a measurement of the total skewness of all the large itemsets.

Definition 2 *Given a database partition D_i, $(1 \leq i \leq n)$, the skewness $TS(D)$ of the partition is defined by $TS(D) = \sum_{X \in L_S} S(X) \times w(X)$, where L_S is the set of all the large itemsets, $w(X) = \frac{X_{.sup}}{\sum_{Y \in I_S} Y_{.sup}}$ is the weight of the support of X over all the large itemsets in L_S, and $S(X)$ is the skewness of X.*

$TS(D)$ has some properties similar to $S(X)$.

- $TS(D) = 0$, when the skewness of all the itemsets are at its minimal value.
- $TS(D) = 1$, when the skewness of all the itemsets are at its maximal value.
- $0 < TS(D) < 1$, in all the other cases.

In Table 4, the skewness $TS(D)$ of the partitions for the three situations have computed. (For illustration purpose, we only have computed $TS(D)$ with respect to the skewness of all the size-1 large itemsets.) Note that we have ignored the small itemsets in the computation of the skewness of a partition. Since the purpose of our task is to investigate the effect of data skewness on candidate sets pruning, and this only involves large itemsets, this restriction would in fact make the metric more relevant to candidate set pruning.

4.2 Workload Balance Metric

Workload balance is a measurement of the distribution of the support clusterings of the large itemsets over the partitions at the processors. Based on the definition of $w(X)$ in Definition 2 and that of $p_X(i)$ in Definition 1, we define $W_i = \sum_{X \in L_s} w(X) \times p_X(i)$ to be the *itemset workload* in a partition D_i, where L_s is the set of all the large itemsets. Intuitively, the workload W_i in partition D_i is the ratio of the total supports of the large itemsets in D_i over all the partitions. Note that $\sum_{i=1}^{n} W_i = 1$.

A partition has high workload balance if W_i are the same for all partitions D_i, $1 \leq i \leq n$. On the other hand, if distribution of W_i over the partitions are very uneven, then the workload balance is low. As has been pointed out, the workload balance has important bearing on the pruning and performance of parallel mining. In parallel to the metric for data skewness, we also define a metric *workload balance factor* to measure the workload balance of a partition, which is based also on entropy.

Definition 3 *For a database partition D_i, $1 \leq i \leq n$, of a database D, the workload balance factor $TB(D)$ of the partition is given by $TB(D) = \frac{-\sum_{i=1}^{n} W_i log(W_i)}{log(n)}$.*

The metric $TB(D)$ has the following properties:

- $TB(D) = 1$, when the workload across all processors are the same;
- $TB(D) = 0$, when the workload is concentrated on one processor;
- $0 < TB(D) < 1$, in all the other cases.

In Table 4, the workload W_i of the first and last cases (Tables 1 and 3) have a high balance, and the values of $TB(D)$ are almost equal to 1. In the second case (Table 2), the workload at processor 2 has been shifted to processor 1, and hence created an unbalance case; the value of $TB(D)$ thus has been reduced to 0.789, which indicates a moderate workload balance.

The data skewness metric and workload balance factor are not independent. Theoretically, each one of them could have values range from 0 and 1. However, some combinations of their values are not admissable. First, let us consider some boundary cases.

Theorem 2 *Let $D_1, D_2, \cdots, D_n$ be a partition of a database D.*

1. *If $TS(D) = 1$, then the admissable values of $TB(D)$ range from 0 to 1. If $TS(D) = 0$, then $TB(D) = 1$.*
2. *If $TB(D) = 1$, then the admissable values of $TS(D)$ range from 0 to 1. If $TB(D) = 0$, then $TS(D) = 1$.*

Proof:

1. By definition $0 \leq TB(D) \leq 1$. What we need to prove is that the boundary cases are admissable when $TS(D) = 1$. $TS(D) = 1$ implies that $S(X) = 1$, for all large itemsets X. Therefore, each large itemset is large at one and only one processor. If all the large itemsets are large at the same processor i, $(1 \leq i \leq n)$, then $W_i = 1$ and $W_k = 0, (1 \leq k \leq n, k \neq i)$. Thus $TB(D) = 0$ is admissable. On the other hand, if each processor has the same number of large itemsets, then $W_i = \frac{1}{n}, (1 \leq i \leq n)$, and $TB(D) = 1$. Furthermore, if $TS(D) = 0$, then $S(X) = 0$ for all large itemsets X. This implies that W_i are the same for all $1 \leq i \leq n$. Hence $TB(D) = 1$.
2. It follows from the first result of this theorem that both $TS(D) = 0$ and $TS(D) = 1$ are admissable when $TB(D) = 1$. Therefore the first part is proved. Furthermore, if $TB(D) = 0$, there exists a partition D_i, $1 \leq i \leq n$, such that $W_i = 1$ and $W_k = 0, (1 \leq k \leq n, k \neq i)$. This implies that all large itemsets are locally large at only D_i. Hence $TS(D) = 1$.

□

Even though, $0 \leq TS(D) \leq 1$ and $0 \leq TB(D) \leq 1$, we have shown in Theorem 2 that not all possible combinations are admissable. In general, the admissable combinations will be a subset of the unit square such as the one in Figure 3. It always contains the two segments $TS(D) = 1$ (S = 1 in Figure 3) and $TB(D) = 1$ (B = 1 in Figure 3), but not the origin, (S = 0, B = 0). After defining the metrics and studying their characteristics, we can validate our observation on the relationship between data skewness, workload balance and candidates pruning effect in our performance studies.

5 Performance Studies

In order to confirm our analysis that the proposed FPM is an efficient algorithm for mining associations in a parallel system, we have implemented all the algorithms on an IBM SP2 and carried out a substantial performance evaluation and comparison.

We have the following three goals in our studies: (1) to verify that FPM is faster than the representative algorithm CD, and confirm that the major performance gain is from the two pruning techniques; (2) to confirm the observation that both data skewness and workload balance are two critical factors in the performance of FPM; (3) to demonstrate that FPM has good parallelism and scalability as a parallel algorithm.

We implemented FPM, its variant FNG, and CD. The IBM SP2 parallel system we used has 32 POWER2 processors (66.7MHz) with 64 MB main memory, running the AIX operating system. Communication between

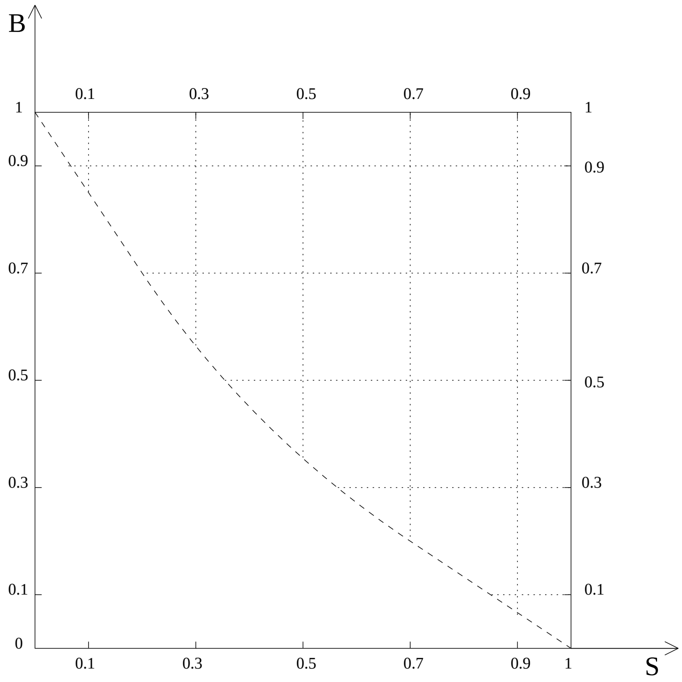


Figure 3: Admissable Combinations of Skewness(S) and Balance(B)

processors are through a high performance switch with an aggregated peak bandwidth of 40 MBps and a latency of about 40 microseconds. Data was allocated to the local disk in each processor, and the database partition on each node is about 100MB in size.

In order to be able to control the experiments to test different data distributions and scenarios, many works [2, 3, 9, 15, 16] in mining association rules have adopted the standard technique introduced in [2] to generate the database. We have enhanced the technique for the generation of database partitions and introduced parameters to control the skewness and workload balance. Table 5 is a list of the parameters used in our synthetic databases. Details of the data generation technique is in the appendix.

D	number of transactions in each partition
T	average size of the transactions
I	average size of the maximal potentially large itemsets
L	number of maximal potentially large itemsets
N	number of items
S	partition skewness
B	workload balance
n	number of partitions

Table 5: Synthetic Database Parameters

5.1 Relative Performance

In order to compare the performance between FMP, FNG, and CD, a databases and twenty data sets have been generated. The data sets generated and their skewness and balance factors are listed in Table 6. The number of partitions in each case is 16 ($n = 16$), and the size of each partition is about 100MB. The name of a partition is denoted by Dx.Ty.Iz.Sr.Bl, where x is the number of transactions in each partitions, y is the average size of the transactions, z is the average size of the itemsets. The two parameters Sr and Bl are two important parameters used to control the skewness and workload balance in the data generation. (The Bl values are listed separately in the table.) In Table 6, we have also computed the measured skewness $TS(D)$ and the balance factor $TB(D)$ of the partitions generated. It is important to note that these measured skewness and workload are very close to the values of the controlled parameters, i.e., the values of S and B are good approximations of values of $TS(D)$ and $TB(D)$. In addition, they cover a wide range of skewness and balanace. Thus, our synthesized data partitions are good simulation of data partitions of various distribution characteristics. We believe this is technically valuable because even real data may not be general enough to cover all possible distributions.

Name	TS(D)						TB(D)					
	B100	B90	B70	B50	B30	B10	B100	B90	B70	B50	B30	B10
D3278K.T5.I2.S90	0.86	0.85	0.86	0.87	0.85	0.88	0.99	0.88	0.69	0.47	0.29	0.08
D3278K.T5.I2.S70	0.74	0.72	0.71	0.72	0.74	-	0.98	0.87	0.68	0.48	0.27	-
D3278K.T5.I2.S50	0.46	0.45	0.47	0.47	-	-	0.98	0.88	0.67	0.47	-	-
D3278K.T5.I2.S30	0.24	0.26	0.25	-	-	-	0.99	0.87	0.66	-	-	-
D3278K.T5.I2.S10	0.07	0.08	-	-	-	-	0.99	0.92	-	-	-	-

Table 6: Attributes of Synthetic Databases

We ran FPM, FNG and CD on the database partitions in Table 6. The minimum support threshold is 0.5%. The improvement of FPM and FNG over CD in response time are recorded in Table 7, and the result is very encouraging. FPM and FNG are consistently faster than CD in all cases. In the following, we analyze the performance gain of FPM and FNG in three aspects: (1) improvement when the workload balance is high, and the skewness varies from high to moderate; (2) improvement when the skewness is high, and the workload balance varies from high to moderate; (3) desirable and undesirable combinations of skewness and workload balance values.

Response Time	FPM/CD						FNG/CD					
Ratio	B100	B90	B70	B50	B30	B10	B100	B90	B70	B50	B30	B10
D3278K.T5.I2.S90	2.10	1.69	1.36	1.23	1.14	1.06	1.79	1.35	1.17	1.10	1.05	1.01
D3278K.T5.I2.S70	2.07	1.41	1.23	1.13	1.06	-	1.43	1.17	1.08	1.05	1.01	-
D3278K.T5.I2.S50	1.88	1.22	1.11	1.06	-	-	1.25	1.10	1.05	1.01	-	-
D3278K.T5.I2.S30	1.55	1.17	1.08	-	-	-	1.08	1.06	1.01	-	-	-
D3278K.T5.I2.S10	1.36	1.09	-	-	-	-	1.03	1.02	-	-	-	-

Table 7: Performance Improvement of FPM and FNG over CD

Figure 4 is the relative performance between FPM, FNG and CD on partitions with different skewness and a high balance value (B = 100). FNG performs much better than CD when the skewness is relatively high ($s > 0.5$). On the

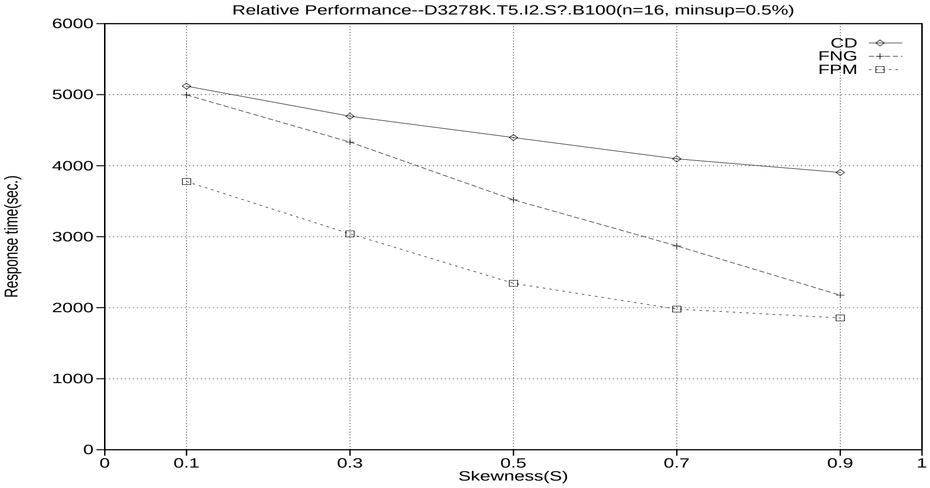


Figure 4: Relative Performance on Databases with High Balance and Different Skewness

other hand, FPM outperforms CD significantly even when the skewness is in the moderate range, $(0.1 \le s \le 0.5)$. When B=90, the result in Table 7 shows that FPM is again much faster than CD. This demonstrates that FPM outperforms CD consistently given a high workload balance and at least a moderate skewness.

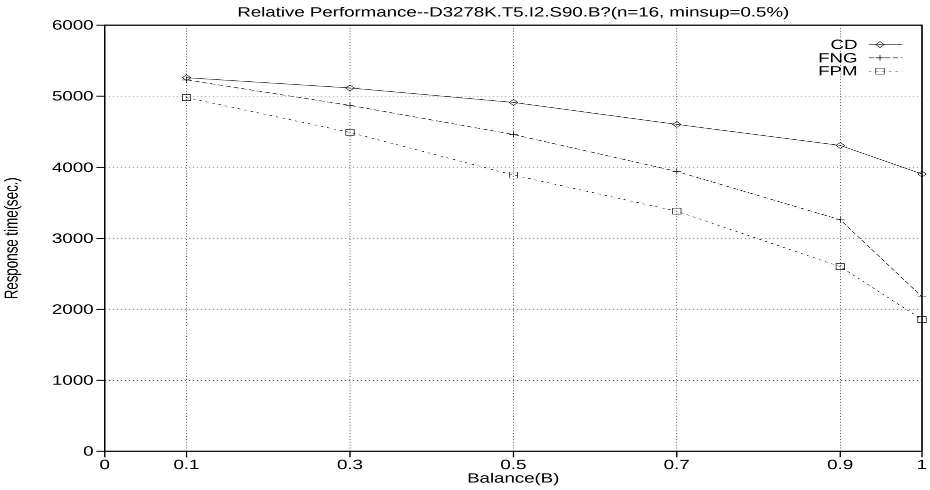


Figure 5: Relative Performance on Databases with High Skewness and Different Workload Balance

Figure 5 is the relative performance given a high skewness (S = 90) and different workload balance. Both FPM and FNG perform much better than CD when the workload balance is relatively high ($B > 0.5$); however, the improvement in the range of moderate balance, $(0.1 \le B \le 0.5)$, is marginal. This confirms our observation that workload balance is an essential requirement. It shows that a high skewness has to accompany by a high workload balance. The effect of a high skewness with a moderate balance may not be as good as that of a high balance with a moderate skewness.

In Figure 6, we vary both the skewness and balance together from a low value range to a high value range. The

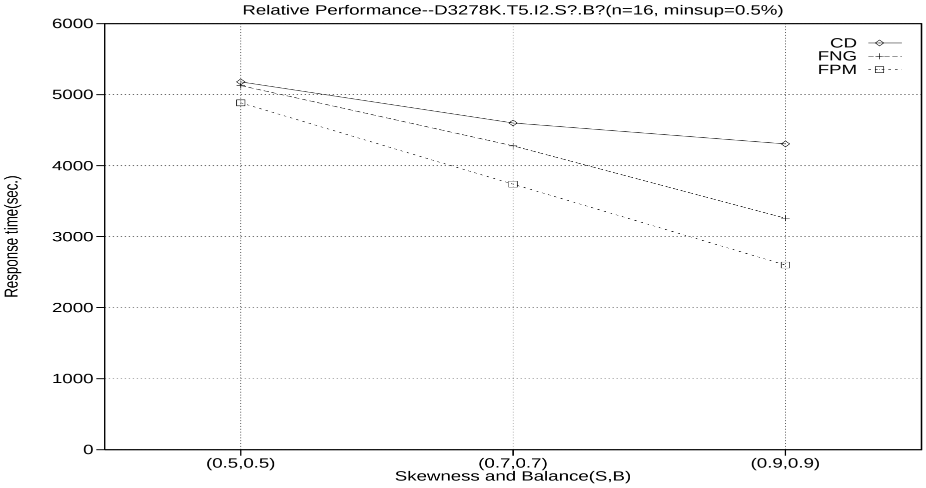


Figure 6: Relative Performance on Databases when both Skewness and Balance change

trend shows that the improvement of FPM over CD increases faster when both values approach the high value range.

Combining the observations in the above three cases together with the results in Table 7, we can divide the area covering all the admissable combinations of skewness and balance in our experiments into four performance regions as shown in Figure 7. Region A is the most favorable region in which the balance is high and the skewness varies from high to moderate. FPM in general is 50% to 100% faster than CD. In region B, the workload balance value has degraded moderately and the skewness remains high; in this case, the gain in FPM over CD is around 50%. Region C covers combinations that have very low workload balance; the gain in FPM falls into a moderate range of about 30%. Region D contains the most undesirable combinations; FPM only has marginal performance gain.

Figure 8 provides us another view to understand the candidates pruning effects. It shows the ratio on the number of candidate sets between FPM (FNG) and CD for the same experiments in Figure 4. The reduction ratios for the runs in the database D3278K.T5.I2.Sr.B100, ($r = 90, 70, 50, 30, 10$), are in the first graph. When the skewness is high, ($s = 0.9$), distributed pruning has a 79.2% of reduction in candidate sets comparing with CD, and global pruning has a 93.9% reduction. When the skewness is low, ($s = 0.1$), distributed pruning only has a 6.6% reduction, but global pruning has a 30.7% reduction. This confirm our observation on the effect of high balance combined with high or moderate skewness.

5.2 Scalability and Parallelism : Speedup and Scaleup

In order to study the efficiency of FPM as a parallel algorithm, we investigate its *speedup* and *scaleup* against CD. Speedup is the reduction in response time vs the number of processors, given that the total size of the database remains unchanged. The more processors are used, the faster the computation should be. The ideal speedup is a linear function on the number of processors. Scaleup is the performance vs the number of processors when the database size is scaled up proportional to the number of proccesors. If the algorithm has high efficiency and low overhead, its performance would maintain uniform when both the number of processors and the size of the database scaled up proportionally.

In the speedup experiment, we execute the algorithms on a fixed size database with various number of processors

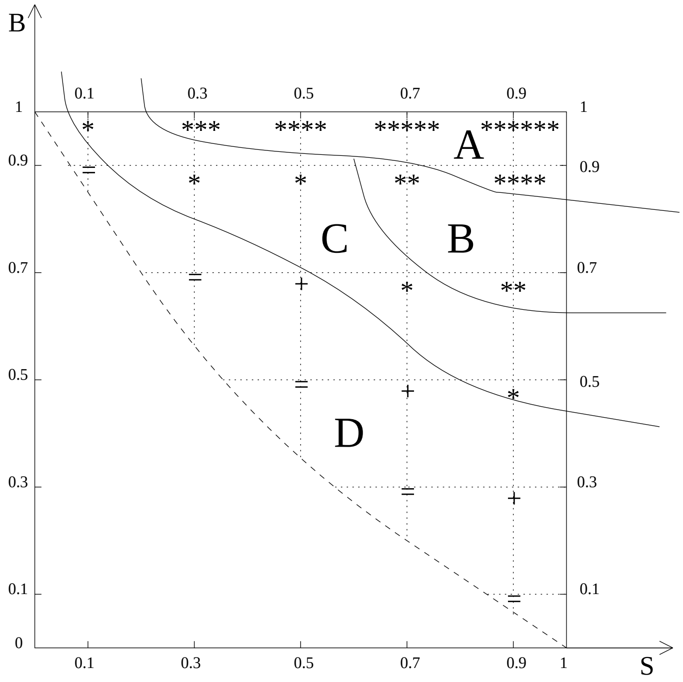


Figure 7: Performance regions (FPM/CD) in the admissible combinations of skewness and workload balance

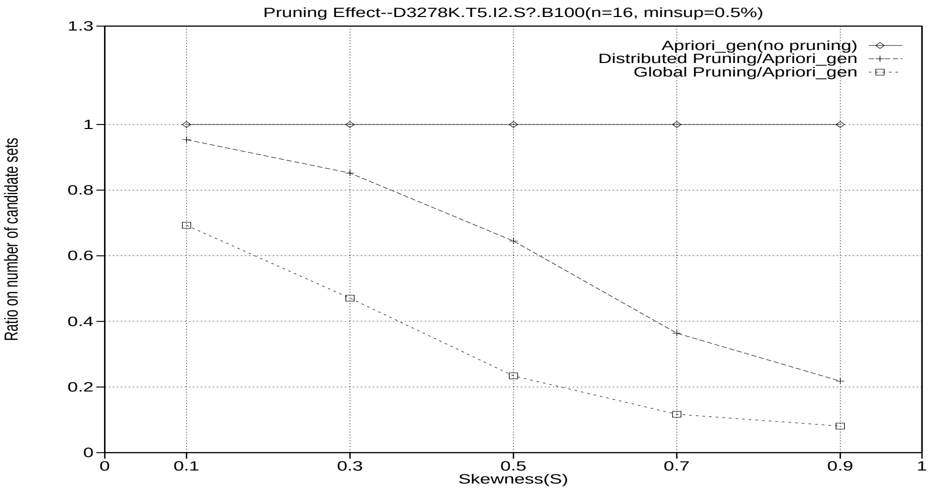
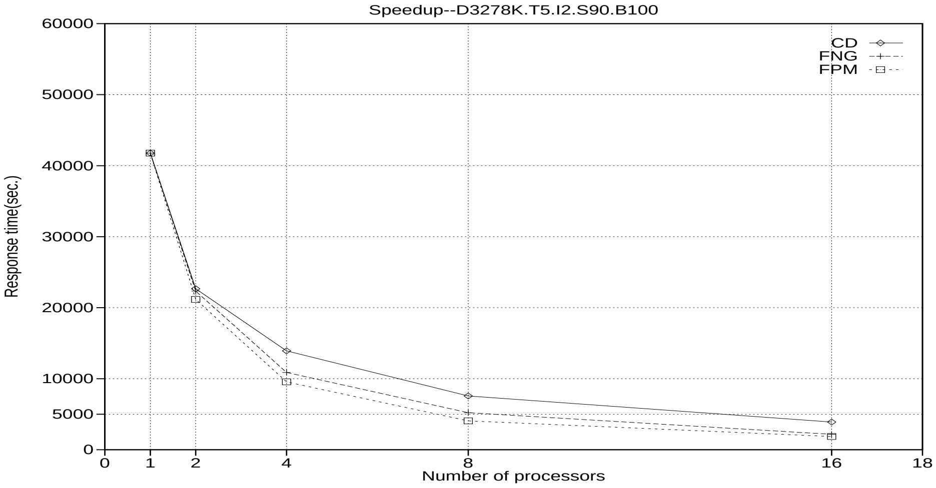


Figure 8: Pruning Effect on Databases in Figure 4



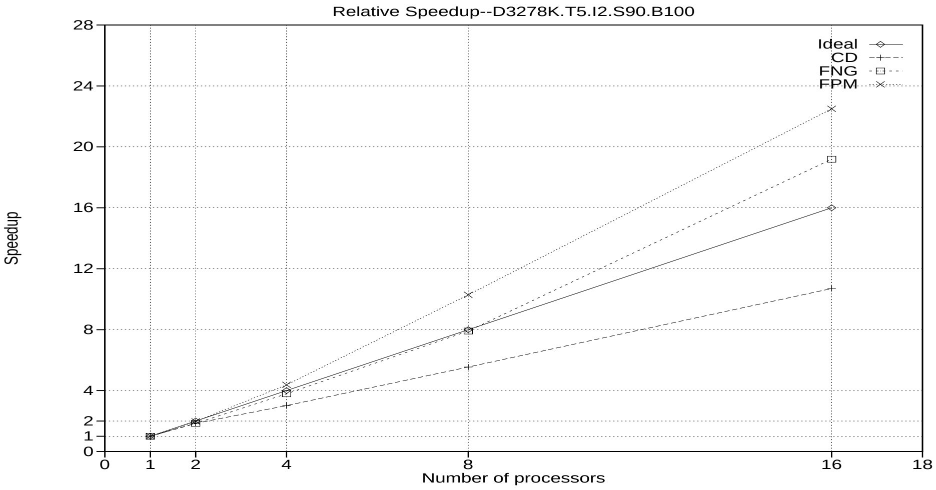


Figure 9: Speedup on a Database (S = 90, B = 100)

and partitions. We selected the database with high skewness and balance as a presentative to perform the study. The database is listed in Table 8. It has a total size of 1.6GB, and was first divided into 16 partitions. Subsequently, we combined the partitions to form databases with 8, 4, 2, and zero partitions.

Figures 9 is the execution times and speedups of FPM, FNG, and CD on the databases. The speedups are also shown in Table 8. FNG had a linear speedup and FPM achieved a remarkable superlinear speedup. The reason behind FPM's superlinear speedup is the increase in the skewness when the number of partitions increases.

In the scaleup experiment, both the database size and the number of processors are scaled up proportionally. The number of processors involved were increased from 1 to 16, and the sizes of the databases were increased correspondingly from 100MB to 1.6GB. Each database were partitioned according to the number of processors such that every partition is maintained at the size of 100MB. We performed the experiment based on the database D3278K.T5.I2.S90.B100, i.e., the databases are generated with the same parameters. Figure 10 shows the result of the experiment. Surprisingly, both FPM and FNG not only can maintain the performance, their response time in

Databases	Speedup of FPM				Speedup of FNG				Speedup of CD			
	n=2	n=4	n=8	n=16	n=2	n=4	n=8	n=16	n=2	n=4	n=8	n=16
D3278K.T5.I2.S90.B100	1.97	4.37	10.29	22.50	1.85	3.79	7.91	19.19	1.85	3.01	5.54	10.70

Table 8: Speedup on five databases with different distribution characteristics

fact had gone down when the database was scaled up. The prime reason for this is the increase in pruing capability when the number of partitions increases.

6 Discussion

To restrict the search of large itemsets in a small set of candidates is essential to the performance of mining association rules. The pruning techniques we proposed are theoretically interesting, because they have effect only in the parallel case but not the serial case. Both distributed and global pruning provide significant amount of pruning power, in particular, when the data distribution is in a favorable situation, i.e. when workload balance is high and the skewness is at least at a moderate level. It is important to study partition techniques that can deliver a good data distribution. Random approaches in general will deliver partitions which have good balance. However, the skewness would be difficult to guarantee together with good workload balance. Clustering technique such as the k-means algorithm [12] will give good skewness. It remains an open problem how to modify clustering technique to generate partitions which have good skewness and also good workload balance.

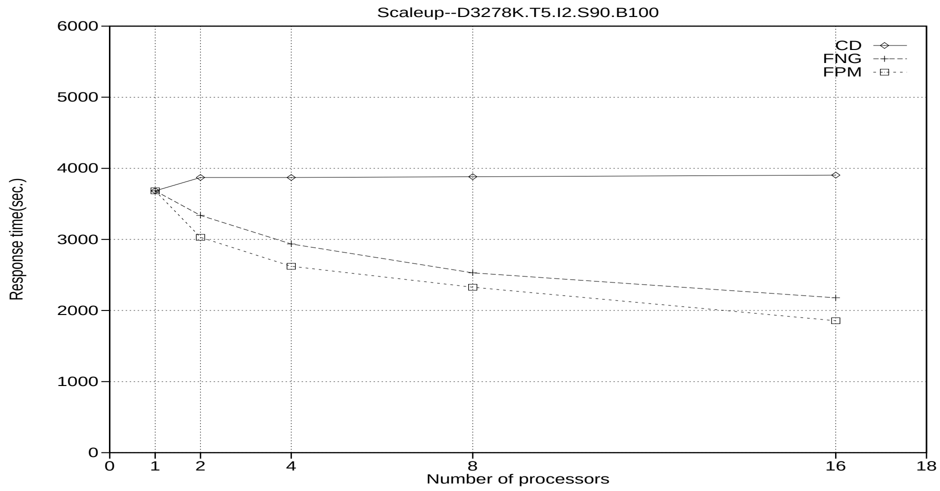
7 Conclusion

A parallel algorithm FPM for mining association rules has been proposed. A performance study carried out on an IBM SP2 shared-nothing memory parallel system shows that FPM consistently outperforms CD. It also has nice scalability in terms of speedup and scaleup. The gain in performance in FPM is due mainly to the pruning techniques incorporated. It was discovered that the effectiveness of the pruning techniques depend highly on the data distribution characteristics, which can be measured by two metrics : data skewness and workload balance. Our analysis and experiment results show that the pruning techniques are very sensitive to workload balance, though good skewness will also have important positive effect. The techniques are very effective in the best case of high balance and high skewness. The combination of high balance and moderate skewness is the second best case. In the worst case of low balance and low skewness, FPM can only deliver the performance close to that of CD. Since mining associations has many interesting applications, important future works would include fine tuning of the proposed parallel techniques on real business cases.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pp. 207–216, May 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pp. 487–499, Santiago, Chile, September 1994.



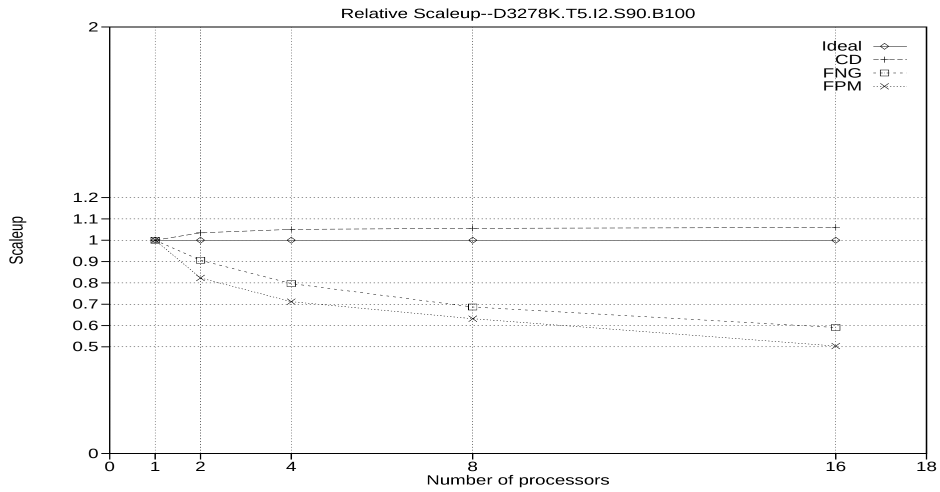


Figure 10: Scaleup on a Database (S = 90, B = 100)

[3] R. Agrawal and J.C. Shafer. Parallel mining of association rules: Design, implementation and experience. Special Issue in Data Mining, *IEEE Trans. on Knowledge and Data Engineering*, IEEE Computer Society, V8, N6, December 1996, pp. 962–969.

[4] S. Brin, R. Motwani, J. Ullman, S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. of 1997 ACM-SIGMOD Int. Conf. On Management of Data*, pages 255–264, Tucson, Arizona, 1997.

[5] T. M. Cover, T. A. Thomas. Elements of information theory. John Wiley & Sons, Inc., 1991.

[6] D. W. Cheung, J. Han, V. T. Ng, A. W. Fu, Y. Fu. A fast distributed algorithm for mining association rules. In *Proc. of 4th Int. Conf. on Parallel and Distributed Information Systems*, pages 31–43, Miami Beach, Florida, December, 1996.

[7] D.W. Cheung, J. Han, V.T. Ng, and C.Y. Wong. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. In *Proc. 1996 IEEE Int. Conf. on Data Engineering*, New Orleans, Louisiana , February 1996.

[8] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1995.

[9] J. Han and Y. Fu. Discovery of Multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, Zurich, Switzerland, pp. 420–431, Sept. 1995.

[10] E. Han, G. Karypis and V. Kumar. Scalable parallel data mining for association rules. In *Proc. of 1997 ACM-SIGMOD Int. Conf. On Management of Data*, 1997

[11] Int'l Business Machines. *Scalable POWERparallel Systems*, GA23-2475-02 edition, February 1995

[12] J. B. MacQueen. Some methods for classification and analysis of multivariate observations, In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, pp.281-297, 1967.

[13] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, May 1994.

[14] R. Ng, L. Lakshmanan, J. Han and A. Pang. Exploratory Mining and Pruning Optimizations of Constrainted Association Rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, Seattle, WH, June 1998.

[15] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data*, pp. 175–186, San Jose, CA, May 1995.

[16] J.S. Park, M.S. Chen, and P.S. Yu. Efficient Parallel Data Mining for Association Rules. In *Proc. 1995 Int. Conf. on Information and Knowledge Management*, Baltimore, MD, Nov 1995.

[17] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pp. 432–444, Zurich, Switzerland, Sept. 1995.

[18] A. Silberschatz, M. Stonebraker, and J. Ullman, Database Research: Achievements and Opportunities Into the 21st Century. In *Report of an NSF Workshop on the Future of Database Systems Research*, May, 1995.

[19] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pp. 407–419, Zurich, Switzerland, Sept. 1995.

[20] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the 5th Int. Conf. on Extending Database Technology*, Avignon, France, 1996.

[21] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD Int. Conf. on Management of Data*, Montreal, Canada, 1996.

[22] T. Shintani, M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *Proc. of 4th Int. Conf. on Parallel and Distributed Information Systems*, 1996

[23] M. J. Zaki, M. Ogihara, S. Parthasarathy, and W. Li, Parallel data mining for association rules on shared-memory multi-processors. *Supercomputing'96*, Pittsburg, PA, Nov 17-22, 1996.

A Synthetic Databases Generation

The synthetic databases used in our experiments are generated using similar techniques introduced in [2]. We have enhanced it to generate data partitions and introduced two parameters to control the skewness and workload balance. Table 5 is a list of the parameters used in our synthetic databases.

The synthetic database partitions are generated from a pool of potentially large itemsets. The first step is to generate the relative weights of these large itemsets. These weights are then broken down into smaller weights with respect to the partitions. Therefore, every itemset in the pool has n weights associated with it. Each one of these weights corresponds to the probability of occurrence of the itemset in a partition. The weight of each itemset in the pool is picked from an exponential distribution with unit mean. A skewness level s for the itemset is then drawn from a normal distribution with mean S and variance 0.1. Following that, n probability values from an exponential distribution with variance equal to s are picked. These n values are normalize so that their sum equals to 1. The skewness of these n probability values are computed according to the skewness metric in Definition 1. We repeat this random process until a set of n values, whose skewness falls into the permitted range of $s \pm 0.02$, is generated. These n probability values are then randomly assigned to the n partitions. Eventually, the weight of the itemset is broken down into n weights by multiplying it with the n probability values. We repeat this process to generate the weights of all the itemsets and their breakdowns. If we use these generated weights as the distribution of the support counts of the itemsets, a high workload balance among the partitions will be resulted because of the randomness in the process. In order to control the balance factor, we redistribute the weights among the partitions starting from this high balance configuration. Firstly, we randomly determine a nondescending order for the workloads of the partitions. We will reshuffle the weights among the partitions according to this order. Secondly, we pick an itemset from the pool randomly, shuffle its n weights among the partitions such that they will be ordered according to the determined workload order. We check the workload balance according to Definition 3, and repeat the process until the balance value is in the permitted range of $B \pm 0.02$. Since the balance value starts at the highest value, it will converge into the expected range. At the end of this step, both the balance value and the workloads of all the partitions are determined.

The second step is to generate the itemsets in the pool. We first divide the N items into n disjoint ranges whose lengths are proportional to the workloads of the corresponding partitions. In order to control the skewness of the itemsets, we regard the i-th ($1 \leq i \leq n$) probability values generated for an itemset in the previous step as the probability of choosing items from the i-th range to put into the itemset. For an itemset whose weights have been determined in the previous step, we first determine the ranges in which the items are picked. These ranges are determined by tossing a n-side weighted coin, where the weight of side i is the i-th probability of the n probability values of the itemset. Once a range has been determined, an item is picked randomly in it. Repeating this procedure, items are picked until the number is equal to the size of the itemset. Some items of the subsequent itemsets are copied from the previous itemset according to the correlation level as in [2], while the remaining items are picked in the same way as in the first itemset.

The last step is to generate the transactions in all the partitions. The technique follows primary the one introduced in [2] with the following modification. After we shuffled the weights to control the workload balance factor, the workload will be different at different partitions. We add a dummy weight to each one of those partitions whose workload is less than the maximum workload to make the workload of all of them equal. The dummy weight corresponds to the weights of the small itemsets in each partition. Therefore, it won't affect the true workload balance. In generating the transactions for partition i, $(1 \leq i \leq n)$, we normalize the i-th weights of all the itemsets so that their sum equals to 1 and use the normalized i-th weight as the probability of occurrence of the associated itemset in partition i. Each transaction in the partition is assigned a series of large itemsets, which are chosen by tossing an (L+1)-side weighted coin, (1 extra side is for the dummy weight), where the weight for a side is the i-th weight of the associated itemset. In the special case that a dummy weight is selected, a dummy which corresponds to small itemsets will be inserted into the transactions. Since the dummy represents small itemsets, it won't be counted into the large itemsets found.