CHAPTER 5

FAST DISTRIBUTED ALGORITHM FOR MINING GENERALIZED ASSOCIATION RULES

Introduction

With the existence of many large transaction databases, the huge amounts of data, the high scalability of distributed systems, and the easy partition and distribution of a centralized database, it is important to investigate efficient methods for distributed mining of generalized association rules. This study discloses some interesting relationships between locally large and globally large itemsets. An interesting fast distributed mining algorithm, FDGM (Fast Distributed Generalized Mining of association rules) has been proposed, which generates a small number of candidate sets and substantially reduces the number of messages to be passed at mining generalized association rules.

In chapter 2, incremental updating techniques for mining generalized association rules in large databases have been presented. Chapters 3 and 4 present fast and efficient algorithm for mining generalized association rules in the sequential environment. All the above algorithms are based on Cumulate algorithm developed by Agrawal and Srikant [AS95].

In this chapter, fast and efficient algorithm for mining generalized association rules in distributed database has been proposed and is based on FDM algorithm. The FDM algorithm is combined with Cumulate algorithm to give fast distributed algorithm called FDGM.

This chapter is organized as follows: In Section 5.1 the previous work related to the distributed environment is presented. Section 5.2 presents the new and fast algorithm for discovering generalized association rules in the distributed environment. In Section 5.3, the performance of FDGM algorithm is compared with the FDM proposed by Cheung et. al. [CNF+96].
5.1 Related Works

Apriori based distributed algorithm called Fast Distributed Mining of association rules (FDM) developed by Cheung et. al. [CHN+96]. The description of FDM algorithm is given below:

Let DB be a database with D transactions. Assume that there are n sites S₁, S₂, ..., Sₙ in a distributed system and the database DB is partitioned over the n sites into {DB₁, DB₂, ..., DBₙ} respectively.

Let Dᵢ, (1 ≤ i ≤ n) be the size of the partition DBᵢ. Let X.Sup and X.Supᵢ be the support counts of an itemset X in DB and DBᵢ respectively. X.Sup is called the global support count and X.Supᵢ is the local support count of X at site Sᵢ. For a given minimum support threshold S, X is globally large if X.Sup ≥ S × D; Similarly, X is locally large at site Sᵢ if X.Supᵢ ≥ S × Dᵢ. Let L denotes all the globally large itemsets in DB and Lₖ denote all globally large k-itemset in L.

The FDM algorithm generates a small set of candidate sets and local pruning candidate sets in each site (Sᵢ) and combines all the locally large itemsets in each site to produce globally large itemsets of whole database DB. Any globally frequent itemset must be locally frequent at a site, the only candidates a site has to consider are the once generated from the once both globally frequent and locally frequent at that site.

FDM suggests three optimizations; local pruning, global pruning, and count polling. Each site generates candidates using the globally large itemsets of site Sᵢ from all sites and assigns a home site for each candidate. Then each site computes the local support for all candidates. Next comes the local-pruning step: remove any itemset X that is not locally frequent, then it must occur at some other sites.

Next each home site requests, for all candidates assigned to it, local counts from all other sites and computes their global support. The home site then broadcasts the global
supports to all other sites. At the end, each site has the globally frequent set, and a new
iteration may begin. Thus, FDM requires far less communication, and local pruning cuts it
down even more.

5.2 Discovering Generalized Association Rules in Distributed Environment

In the following sections, the problem of mining generalized association rules in the
distributed environment is presented. The following section concentrates on the problem
description followed by the novel and efficient algorithm for mining generalized association
rules in the distributed environment.

Problem Description

Let DBj (1 ≤ i ≤ n) be a partitioned database located at n sites S1, S2, ..., Sn with their
sizes as DB1, DB2, ..., DBn respectively.

Let the size of DB and the partitions DBj be D and Dj, respectively. For a given
itemset X, let X.Sup and X.Supi be the respective support counts of X in DB and DBj. It is
called X.Sup. The global support count and X.Supi is the local support count of X at site Sj.
For a given minimum support S, X is globally large if X.Sup > S x D; accordingly, X is
locally large at site Sj, if X.Supi > S x Dj. Let L is used to denote all the globally large
itemsets in DB and L* is used to denote all globally large k-itemsets in L. The problem of
mining generalized association rules in distributed database DB can be reduced to find all
globally large itemsets.

Generation of Candidate Sets

There is an important relationship between large itemsets and the sites in a distributed
database: every globally large itemset must be locally large at some site(s). If an itemset X is
both globally large and locally large at a site Si, X is called globally large at site Si. The set
of a globally large itemsets at a site will form the basis for the site to generate its own
candidate sets.
The following two properties can be easily found from the locally large and globally large itemsets.

a) if an itemset \(X\) is locally large at site \(S_i\), then all of its subsets are also locally large at site \(S_i\).

b) if an itemset \(X\) is globally large at site \(S_i\), then all of its subsets are also globally large at site \(S_i\).

Following is an important result based on which an effective technique for candidate sets generation in the distributed case has been developed.

Table 5.1 Notation Table

<table>
<thead>
<tr>
<th>D</th>
<th>Number of transactions in DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Support threshold</td>
</tr>
<tr>
<td>(L_{(k)})</td>
<td>Globally large itemsets</td>
</tr>
<tr>
<td>(CA_{(k)})</td>
<td>Candidate sets generated from (L_{(k)})</td>
</tr>
<tr>
<td>(X.sup)</td>
<td>Global support count of (X)</td>
</tr>
<tr>
<td>(D_i)</td>
<td>Number of Transactions in DB_i</td>
</tr>
<tr>
<td>(GL_{(i)})</td>
<td>Globally large itemsets at (S_i)</td>
</tr>
<tr>
<td>(CG_{(ik)})</td>
<td>Candidate sets generated from (GL_{(i(k-1))})</td>
</tr>
<tr>
<td>(LL_{(ik)})</td>
<td>Locally large (k)-itemsets in (CG_{(ik)})</td>
</tr>
<tr>
<td>(X.sup_i)</td>
<td>Local support count of (X) at (S_i)</td>
</tr>
</tbody>
</table>

Lemma 5.1: If an itemset \(X\) is globally large at site \(S_i\) \((1 \leq i \leq n)\), then \(X\) and all its subsets are also globally large at site \(S_i\).

Proof: If \(X\) is globally large at any site, then \(X.Sup_i > S \times D_i\) \((1 \leq i \leq n)\). Therefore, \(X.Sup_i < S \times D_i\) \((1 \leq i \leq n)\), and \(X\) cannot be globally large. And \(X\) must be locally large at some site \(S_i\), and hence \(X\) is globally large at site \(S_i\). All the subsets of \(X\) must also be globally large at site \(S_i\).
Using the above result, if an itemset \( X \) is globally large, there exists a site \( S_i (1 \leq i \leq n) \), such that \( X \) and all its subsets are globally large at site \( S_i \). \( \text{GL}_i \) is used to denote the set of globally large itemsets at site \( S_i \), and \( \text{GL}(k) \) is used to denote the set of globally large \( k \)-itemsets at site \( S_i \). If \( X \in L_{(k)} \), then, there exists a site \( S_i \), such that all its size\((k-1)\) subsets are globally large at site \( S_i \), (i.e., they belong to \( \text{GL}_{i(k-1)} \)).

The straightforward adaptation of Cumulate algorithm is the set of candidate sets at the \( k^{th} \) iteration denoted by \( \text{CA}_{(k)} \), which stands for size-\( k \) candidate sets from Cumulate algorithm which would generate the large itemsets of size \((k-1)\) by applying the Candidate-gen function on \( L_{(k-1)} \). (i.e., \( \text{CA}_{(k)} = \text{candidate-gen}(L_{(k-1)}) \)).

At each site \( S_i \), let \( \text{CG}_{i(k)} \) be the set of candidates sets generated by applying candidate-gen on \( \text{GL}_{i(k-1)} \), i.e., \( \text{CG}_{i(k)} = \text{candidate-gen}(\text{GL}_{i(k-1)}) \), where \( \text{CG} \) stands for candidate sets generated from globally large itemsets. So, \( \text{CG}_{i(k)} \) is generated from \( \text{GL}_{i(k-1)} \). Since \( \text{GL}_{i(k-1)} \subseteq L_{(k-1)} \), \( \text{CG}_{i(k)} \) is a subset of \( \text{CA}_{(k)} \). In the following, \( \text{CG}_{(k)} \) is used to denote the set \( \Sigma \text{CG}_{i(k)} \).

**Theorem 5.1** For all \( k > 1 \), the set of all small \( k \)-itemsets \( L_k \) is a superset of \( \text{CG}_k = \cup \text{CG}_{ik} \), where \( \text{CG}_{ik} = \text{candidate-gen}(\text{GL}_{i(k-1)}) \). Here \( \text{CG}_k \) denote the globally candidate sets of size-\( k \).

**Proof:** Let \( X \in L_k \), there exists a site \( S_i \), (1 \( \leq \) i \( \leq \) n), such that all the size\((k-1)\) subsets of \( X \) are globally large at site \( S_i \). Hence \( X \in \text{CG}_{ik} \). Therefore, \( L_k = \cup \text{CG}_{ij} \) (1 \( \leq \) i \( \leq \) n) = candidate-gen (GL_{i(k-1)}).

By using theorem 5.1, for every \( k > 1 \), the set of all large \( k \)-itemsets \( L_k \) is a subset of \( \text{CH} = \cup \text{CH}_k \), where \( \text{CH}_k \) = Candidate-gen (GL_{(k-1)}). Hence \( \text{CH}_k \) is a set of candidate sets for the size-\( k \) large itemsets. Theorem 5.1 indicates that \( \text{CG}_{(k)} \), which is a subset of \( \text{CA}_{(k)} \) and could be much smaller than \( \text{CA}_{(k)} \), can be taken as a set of candidate sets for the size-\( k \) large item sets. The difference between the two sets \( \text{CA}_{(k)} \) and \( \text{CG}_{(k)} \), depends on the distribution of the itemsets. First the set of candidate sets \( \text{CG}_{i(k)} \) can be generated locally at each site \( S_i \) at the \( k^{th} \) iteration. After exchanging the support count, the globally large itemsets \( \text{GL}_{i(k)} \) can be
found at the end that iteration. Based on \( GL_i(k) \), the candidate sets at \( S_i \) for the \((k+1)\)th iteration can then be generated. According to the performance study in section 5.6, using this approach, the number of candidate sets generated can be substantially reduced to about 10 to 20% of that generated in Cumulate algorithm. Illustration 5.1 shows the effectiveness of the reduction of candidate sets.

**Illustration 5.1:** Let us assume that there are 3 sites in a distributed system, which partitions the DB into \( DB_1 \), \( DB_2 \) and \( DB_3 \). Suppose the set of large-1 itemsets (computed at first iteration) \( L_{(1)} = \{ G, C, E, A, B \} \), in which the itemset \( \{ G, C, A, B \} \) are locally large site \( S_1 \), the itemset \( \{ C, A, E, B \} \) are locally large at site \( S_2 \) and itemset \( \{ G, C, E, B \} \) are locally large at site \( S_3 \). Therefore \( GL_{1(1)} = \{ G, C, E, A, B \}, GL_{2(1)} = \{ G, C, A, B \} \) and \( GL_{3(1)} = \{ C, A, E, B \} \) are the globally large size-1 itemsets.

Based on Theorem 5.1, the set of size-2 candidate sets at site \( S_1 \) is \( CG_{1(2)} \), where
\[
CG_{1(2)} = \text{Candidate-gen}(GL_{1(1)}) = \{(G, C), (G, A), (G, B), (C, A), (C, B), (A, B)\}. 
\]
After that any candidate in \( C_2 \) that consists of an item and its ancestor is deleted. It should be noted that it is not necessary to be counted as an itemset, which contains both an item and its ancestor. Ancestors in \( T \) that are not present in any of the candidates are removed. Hence \( CG_{1(2)} = \{(G, B), (C, B), (A, B)\} \).

Similarly \( CG_{2(2)} = \{(C, A), (C, E), (C, B), (A, B), (A, E)\} \) and the candidate itemsets that contain itemsets and their ancestors are removed. Ancestors in \( T \) that are not present in any of the candidates are dropped. Then the site \( S_2 \) is \( CG_{2(2)} = \{(C, E), (C, B), (A, B), (A, E)\} \) and \( CG_{3(2)} = \{(C, A), (C, E), (C, B), (A, B), (A, E)\} \) and after pruning the candidates in the site \( S_3 \) is \( CG_{3(2)} = \{(C, E), (C, B), (A, E), (A, B)\} \).

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 of 5 candidate itemsets. However if Candidate-gen is applied to \( L_{(1)} \), the set of candidate sets \( CA_{(2)} = \text{Candidate-gen}(L_{(1)}) \) would be of 10 candidate sets. This shows that it is very efficient and effective to reduce the candidate sets.
Local Pruning of Candidate Set

Based on the above discussion, one can usually generate in a distributed environment a much smaller set of candidate sets than by the direct application of the Cumulate algorithm.

When the set of candidate set \( CG(k) \) is generated to find the globally large itemsets, the support counts of the candidate sets must be exchanged among all the sites. One should notice that some candidate sets in \( CG(k) \) can be pruned by a local pruning technique before count exchange starts. The general idea is that at each site \( S_i \), if a candidate set \( X \in CG_i(k) \) is not locally large at site \( S_i \), \( S_i \) need not be found. This is because in this case, either \( X \) is small (not globally large), or it is locally large at some other site, and hence only the site(s) at which \( X \) is locally large need to be responsible to fit the global support count of \( X \). Therefore, in order to compute all the large k-itemsets at each site \( S_i \), the candidate sets can be confined to only the sets \( X \in CG_i(k) \) which are locally large at site \( S_i \). For convenience, \( LL_i(k) \) is used to denote those candidate sets in \( CG_i(k) \), which are locally large at site \( S_i \).

Based on the above discussion, for all iterations from first iteration to the k-th iteration, the globally large k-itemsets can be computed at each site \( S_i \) according to the following procedure:

a) **Candidate Sets Generation:** Generate the candidate sets \( CG_i(k) \) based on the globally large itemsets found at site \( S_i \) at the (k-1)\(^{th} \) iteration using the formula, \( CG_i(k) = \text{Candidate-gen}(GL_j(k,i)) \).

b) **Local Pruning:** For each \( X \in CG_i(k) \), the partition \( DB_i \) is scanned, the candidate itemsets that contain an itemsets and their ancestors are removed and the local support count \( X.\sup_i \) is computed. If \( X \) is not locally large at site \( S_i \), it is excluded from the candidate sets \( LL_i(k) \). But this pruning removes only \( X \) from the candidate set at site \( S_i \) and \( X \) could still be a candidate set at some other site.

c) **Support Count Exchange:** The candidate sets in \( LL_i(k) \) are broadcast to other sites to collect support counts. The global support counts are computed and all the
globally large k-itemsets in sites $S_i$ are found out. If any candidate sets in $LL_{j(k)}$ that contain the itemsets and their ancestors, then those itemsets are deleted.

d) **Broadcast mining results:** The computed globally large k-itemsets are broadcast to all the other sites. For clarity, notations are used as listed in Table 5.1.

To illustrate the above procedure, it is continued in working illustration 5.1, as below:

**Illustration 5.2:** In illustration 5.1, it is assumed that the database has 120 transactions and each of the 3 partitions has 40 transactions. It is also assumed that the support threshold $S$ is 10%. Moreover, as illustrated in 5.1, in the second iteration, the candidate sets generated at site $S_1$ are $CG_{1(2)} = \{(G, C), (G, A), (G, B), (C, A), (C, B), (A, B)\}$; at site $S_2$ $CG_{2(2)} = \{(C, A), (C, E), (C, B), (A, B), (A, E)\}$ and at site $S_3$, $CG_{3(2)} = CG_{3(2)} = \{(C, A), (C, E), (C, B), (A, B), (A, E)\}$.

In order to compute the large-2 itemsets, FDGM algorithm first generates large itemsets of size-2 itemsets. If an itemset pair contains an item and its ancestor, then the pair is removed from the set in site $S_1$. After deleting the candidate sets and their ancestors in $S_1$, $CG_{1(2)} = \{(G, B), (C, B), (A, B)\}$; at site $S_2$ $CG_{2(2)} = \{(C, E), (C, B), (A, B), (A, E)\}$ and at $S_3$, $CG_{3(2)} = \{(C, E), (C, B), (A, E), (A, B)\}$ are found.

After that, FDGM computes the local support counts at each site $S_i$. The results are stored in Table 5.2. For example, the candidate sets at site $S_1$ after deleting the candidate itemsets and their ancestors at site $S_1$ the remaining candidate sets are listed in the first column and the local support count are listed in the second column.

From Table 5.2, it is found that $(G, B)$ is globally large. So, at site $S_1$, the candidate set $(G, B)$ is deleted. But $(C, B)$ and $(A, B)$ have some minimum local support counts, and they retained in site $S_1$. Hence, the locally large itemsets $LL_{1(2)} = \{(C, B), (A, B)\}$. Similarly $LL_{2(2)} = \{(C, E), (A, B), (A, E)\}$ and $LL_{3(2)} = \{(C, E), (C, B), (A, E)\}$ are also locally large itemsets. After completing local pruning of all the three sites $S_1, S_2$ and $S_3$, the number of

100
size-2 candidate itemsets is reduced to 4, which is less than half of the original size. After completing local pruning, each site broadcasts messages containing all the remaining candidate itemsets to the other sites to collect their support counts.

Table 5.2 Locally Large Itemsets

<table>
<thead>
<tr>
<th>Locally Large Candidates</th>
<th>Broadcast request from</th>
<th>X.sup_1</th>
<th>X.sup_2</th>
<th>X.sup_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A, B)</td>
<td>S_1</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>A, E</td>
<td>S_2, S_3</td>
<td></td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>(C, B)</td>
<td>S_1, S_2, S_3</td>
<td>10</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>(C, E)</td>
<td>S_1, S_2</td>
<td></td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

The result of this count support exchange is stored in Table 5.3. The Support count for (A, B) is broadcast from S_1 to site S_2 and S_3, and the counts sent back are stored at site S_1 as in the second row of Table 5.3. The other rows record similar count exchange activities at the other sites. At the end of the iteration, site S_1 finds out (A, E), (C, B), (C, E) to be globally large because (A, E).Sup = (10 + 8) > 10 % x 120 = 18 > 12; (A, B).Sup = (10 + 2 + 6 ) > 10 % x 120 = 18 > 12 and (C, E).Sup = (4 + 10) > 10 % x 120 = 14 > 12. Hence, the globally large-2 itemsets at site S_1 is GL_{1(2)} ={(C, B), (A, B)}. Similarly GL_{2(2)}={(C,E), (C, B), (A, E)}. After that all the sites return the large 2-itemsets L_{(2)} = {(C, B), (A, B), (C, E), (A, E)}. From Table 5.3, it is known that {(A, E), (C, B), (C, E)} could be large at more than one site. Hence, {(A, E), (C, B), (C, E)} are found to be locally large.
If the size of all candidate itemsets is smaller than the size of the memory of each site, all are done in its site. By dividing the transaction databases over all the sites, the transaction data can be read and candidate itemsets can be counted in mining association rules.

When the candidate itemsets do not fit into the local memory of a site, the candidate itemsets are divided into segments, each of which fits in the memory site of a node in a site $S_i$. Then each segment of the transaction database is scanned. Thus such repetitive scanning of transaction database incurs the excessive I/O's and reduces the performance significantly.

For simplicity, it is assumed that the size of the candidate itemsets is smaller than the size of local memory of the each site $S_i$. Then generalized association rules in distributed databases are generated globally.

Each Site Works as below:

a) Generate the candidate itemsets: Each site generates the candidate $k$-itemsets $C_k$ using the large $(k-1)$-itemsets $L_{k-1}$. If $(k = 2)$, the candidate itemsets that contain itemsets and their ancestors are deleted. Then, $C_k$ is inserted into the hash table, and any ancestors in $T$ that are not present in any of the candidates in $C_k$ are deleted.

b) Scan the transaction database and count the support count value: Each site reads the transaction database and generates extended transaction $t'$. (i.e., by adding all the ancestors of the items in a transaction $t$ that are present in $T$ to $t'$). The support count of all the candidates in $C_k$ that are contained in $t'$ is added.

c) Determine the large itemsets: After reading all the transaction data in all sites, the support count is found and checked to determine whether the minimum support condition is satisfied or not.
d) If the candidate $k$-itemsets are empty the algorithm terminates. Otherwise the site $S_i$ broadcasts large itemsets to all the sites. $k = k + 1$ and goes to step number 1.

The FDGM Algorithm

In the following section, the proposed distributed algorithm FDGM is presented in a detailed manner.

Algorithm Fast Distributed Generalized Association Rules Mining (FDGM)

Input: (1) $DB_i$: The database partition at each site with equal size, say $D_i$;  
(2) $s$: the minimum support threshold; both used at each site $S_i$ ($i = 1, \ldots, n$).

Output: $L$: the set of large itemsets in $DB$ at all sites.

Method: The following segments are distributed at all sites $S_i$ initially $k = 1$, where $k$ is the counter for initialization; the loop terminates if ($L_k = \emptyset$) or a set of candidate sets is null.

Compute $T^*$, the set of ancestors of each item, from $T$.

\[/* \text{local pruning} */\]

if $k = 1$ then
scan $DB_i$ to compute $T_{i(1)}$;
\[/* T_{i(1)} \text{ is an array of all size 1 large itemsets in } DB_i \text{ and its local support count in site } S_i */\]
if $k \geq 2$
$C_k = \text{the candidates of size } k \text{ generated from } L_{k-1}$.
If ($k = 2$) then
Delete the candidates that contain itemsets and their ancestors.
Delete any ancestors in $T_{i(1)}$ that are not present in $C_k$
For all transactions $t \in D_i$ do
Begin
For each item $x \in t$ do
  Add all ancestors of $x$ in $T_{i(1)}$ to $t$
Remove any duplicates from $t$.
Increment the count of all candidates in $C_k$ that are contained in $t$
End;
For all $X \in T_i(k)$ do
   If $X.sup_i \geq s \times D_i$ then
      For $j = 1$ to $n$ do
         If polling-site($X$) = $S_j$ then
            Add $\langle X, X.sup_i \rangle$ in $LL_{i,j}(k)$;
/* Send candidate sets to polling sites */
   for $j = 1, ..., n$ do
      send $LL_{i,j}(k)$ to site $S_j$;
/* Receive Candidate sets as a Polling site */
   for $j = 1, ..., n$ do {
      receive $LL_{j,i}(k)$
      for all $X \in LL_{j,i}(k)$ do {
         store $X$ in $LP_{i}(k)$;
         update $X$ large sites in $LP_{i}(k)$ to record the sites at which are locally large;
      }
   }/* Send Polling Requests as a Polling Site to Collect Support Counts */
   for all $X \in LP_{i}(k)$ do {
      Broadcast polling requests for $X$ to the sites $S_j$, where $S_j \in X$ large sites;
   }/* Compute Global Support Counts and Globally large itemsets */
   For all $X \in LP_{k}(k)$ do {
      $X.sup = \sum_{i=1,n} X.sup_i$;
      If $X.sup \geq s \times D$ then insert $X$ into $CG_{i}(k)$;
      /* filter out the globally large $k$-itemsets; */
      Broadcast $LL_{i}(k)$;
      Receive $LL_{j}(k)$ from all other sites $S_j$, $(j \neq k)$;
      return $L_k = \cup_{i=1,n} CG_{i}(k)$.

Figure 5.1 FDGM - Fast Distributed Generalized Association rules Mining
In Figure 5.1, every site $S_i$ initially acts as a host. The candidate itemsets are generated and it is found to be locally large itemsets. Later, it becomes a polling site to serve the requests from other sites. Subsequently, it changes its status to a remote site to supply local support frequencies to other sites. The corresponding steps in the above algorithm for these different activities are grouped as follows:

a) Home site: it generates candidate sets and submits them to polling sites.

b) Polling site: receives candidate sets and sends polling requests.

c) Remote site: returns support counts to polling sites.

d) Polling site: receives support counts and finds large itemsets.

e) Home site: receives large itemsets.

Thus, FDGM algorithm is used to generate large frequent itemsets in the distributed environment and efficiently produce large itemsets.

5.3 Performance Evaluation

The performance study has been done to compute FDGM algorithm with Cumulate algorithm. As in Table 5.1, and Figures 5.2 and 5.3 the experimental results show that the number of candidate sets found in the proposed algorithm at each site is between 10-25% of the single machine. The execution time of FDGM is between 50-60% of that of single machine. The reduction in the number of candidate sets and message size in FDGM is significant. The performance of FDGM against Cumulate in a large database is also compared. In that case, the response time of FDGM is only about 25% longer than $(1/n)$ of the response time of Cumulate algorithm where $n$ is the number of sites. This is a very ideal speed-up. In terms of total execution time, FDGM is very close to Cumulate algorithm.
Table 5.4 Candidate set reduction for a set of transactions for 4 nodes.

<table>
<thead>
<tr>
<th>Minimum Support</th>
<th>5%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
<th>25%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candidate sets</td>
<td>627</td>
<td>375</td>
<td>165</td>
<td>58</td>
<td>45</td>
</tr>
<tr>
<td>New Candidate sets</td>
<td>442</td>
<td>259</td>
<td>132</td>
<td>120</td>
<td>44</td>
</tr>
<tr>
<td>Execution time</td>
<td>184</td>
<td>115</td>
<td>45</td>
<td>34</td>
<td>14</td>
</tr>
</tbody>
</table>

Figure 5.2 Message Size Reduction

Figure 5.3 Execution Time (n = 1, 2, 3, 4)