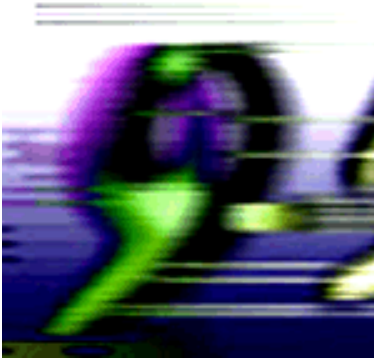


ODAM: An Optimized Distributed Association Rule Mining Algorithm

Mafruz Zaman Ashrafi, *Monash University*
David Taniar, *Monash University*
Kate Smith, *Monash University*



Association rule mining is an active data mining research area. However, most ARM algorithms cater to a centralized environment. In contrast to previous ARM algorithms, ODAM is a distributed algorithm for geographically distributed data sets that reduces communication costs.

Modern organizations are geographically distributed. Typically, each site locally stores its ever-increasing amount of day-to-day data. Using centralized data mining to discover useful patterns in such organizations' data isn't always feasible because merging data sets from different sites into a centralized site incurs huge network communication costs. Data from these organizations are not only distributed over various locations but also vertically fragmented, making it difficult if not impossible to combine them in a central location. *Distributed data mining* has thus emerged as an active subarea of data mining research.

A significant area of data mining research is *association rule mining*. Unfortunately, most ARM algorithms¹⁻⁹ focus on a sequential or centralized environment where no external communication is required. *Distributed ARM* algorithms, on the other hand, aim to generate rules from different data sets spread over various geographical sites; hence, they require external communications throughout the entire process. DARM algorithms must reduce communication costs so that generating global association rules costs less than combining the participating sites' data sets into a centralized site. However, most DARM algorithms don't have an efficient message optimization technique, so they exchange numerous messages during the mining process.

We have developed a distributed algorithm, called Optimized Distributed Association Mining, for geographically distributed data sets. ODAM generates support counts of candidate itemsets quicker than other DARM algorithms and reduces the size of average transactions, data sets, and message exchanges.

ASSOCIATION RULE MINING ALGORITHMS

Many parallel or distributed ARM algorithms exist in the data mining literature.^{2,10,11} However, most were designed for shared memory parallel environments. Based on the nature and implementation of each algorithm, we can divide the existing algorithms into two groups: parallel ARM and DARM.

Parallel

We can categorize parallel ARM algorithms as *data-parallelism* or *task-parallelism* algorithms. In the former, the algorithms partition the data sets among different nodes; in the latter, each site performs the task independently but must access the entire data set.¹²

The Count Distribution (CD) algorithm is a simple data-parallelism algorithm.² It uses the sequential Apriori algorithm in a parallel environment and assumes data sets are horizontally partitioned among different sites. The CD algorithm's main advantage is that it doesn't exchange data tuples between processors—it only exchanges the counts. In the first scan, each processor generates its local candidate itemset depending on the items present in its local partition. The algorithm obtains global counts by exchanging local counts with all other processors. The algorithm's communication overhead is $O(|C| \cdot n)$ at each phase, where $|C|$ and n are the size of candidate itemsets and number of data sets, respectively.

Data Distribution is a task-parallelism-based algorithm that partitions the candidate itemsets among the processors.² Each processor is responsible for computing the counts of its locally stored subset of the candidate itemsets for all the transactions in the database. Each processor must scan the portions of the

transactions assigned to other processors as well as its locally stored portion of the transactions. It thus suffers from high communication overhead and performs poorly when compared with CD.

Candidate Distribution partitions the candidates during iterations, so that each processor can generate disjoint candidates independently.² At the same time, it selectively replicates the database so that a processor can generate global counts independently. Candidate Distribution performs worse than CD.

The Common Candidate Partitioned Database uses a data-parallel approach in a shared-memory architecture.⁹ The algorithm partitions the database logically into equal-sized chunks. Each processor generates a disjoint candidate subset, leading to good computational division.

The PEAR algorithm¹³ is based on the sequential SEAR algorithm. The SEAR algorithm works exactly like the Apriori algorithm but uses a prefix tree rather than a hash tree, improving its performance.

Masaru Kitsuregawa and his colleagues have presented four algorithms known as Non Partitioned Apriori, Simply Partitioned Apriori, Hash Partitioned Apriori, and Hash Partitioned Apriori with Extremely Large Itemset Duplication.¹⁴ NPA copies the candidate itemsets into all processors, so each processor can work independently. The coordinator processor then gathers the final statistics and examines the support counts. The SPA candidate itemsets are partitioned among different processors, and each processor broadcasts its local transaction to all processors. HPA is similar to SPA except that it reduces the broadcasting cost by using hash functions such as *hash join*. The HPA-ELD algorithm uses memory more efficiently when candidate itemsets are smaller than the available system memory.

Distributed

DARM discovers rules from various geographically distributed data sets. However, the network connection between those data sets isn't as fast as in a parallel environment, so distributed mining usually aims to minimize communication costs.

Researchers proposed the Fast Distributed Mining algorithm to mine rules from distributed data sets partitioned among different sites.^{3,10} In each site, FDM finds the local support counts and prunes all infrequent local support counts. After completing local pruning, each site broadcasts messages containing all the remaining candidate sets to all other sites to request their support counts. It then decides whether large itemsets are globally frequent and generates the candidate itemsets from those globally frequent itemsets.

FDM's main advantage over CD is that it reduces the communication overhead to $O(|C_p| * n)$, where $|C_p|$ and n are potentially large candidate itemsets and the number of sites, respectively. FDM generates fewer candidate itemsets compared to CD, when the number of disjoint candidate itemsets among various sites is large. However, we can only achieve this when different sites have nonhomogeneous

data sets. FDM's message optimization techniques require some functions to determine the polling site, which could cause extra computational cost when each site has numerous local frequent itemsets. Furthermore, each polling site must send a request to remote sites other than the originator site to find an itemset's global support counts, increasing message size when numerous remote sites exist.

Recently, Assaf Schuster and his colleagues proposed the Distributed Decision Miner,¹⁴ which reduces communication overhead to $O(Pr_{above} * |C| * n)$, where Pr_{above} is the probability of a candidate itemset that has support greater than the support threshold. It generates only those rules that have confidence above the threshold level without generating a rule's exact confidence, therefore considering all rules above the confidence threshold as being the same. However, ARM is an iterative process, and it's hard for an algorithm to guess a priori how many rules might satisfy a given level of support or confidence.¹⁵ Furthermore, the final rule model this approach generates won't be identical at different sites because it generates rules using an itemset's partial support count.

DESIGN RATIONALE

Unlike other algorithms, ODAM offers better performance by minimizing candidate itemset generation costs. It achieves this by focusing on two major DARM issues—*communication* and *synchronization*. Communication is one of the most important DARM objectives. DARM algorithms will perform better if we can reduce communication (for example, message exchange size) costs. Synchronization forces each participating site to wait a certain period until globally frequent itemset generation completes. Each site will wait longer if computing support counts takes more time. Hence, we reduce the computation time of candidate itemsets' support counts.

To reduce communication costs, we highlight several message optimization techniques.^{2,3,11} Based on our discussion of ARM algorithms and on the message exchange method, we can divide the message optimization techniques into two methods—direct and indirect support counts exchange. Each method has different aims, expectations, advantages, and disadvantages. For example, the first method exchanges each candidate itemset's support count to generate globally frequent itemsets of that pass (CD and FDM are examples of this approach). All sites share a common globally frequent itemset with identical support counts, so rules that are generated at different participating sites have identical confidence. This approach focuses on a rule's exactness and correctness.

The second method intends to discover association rules that have confidence above the threshold level (the Distributed Decision Miner algorithm is an example). It aims to reduce communication costs, so it doesn't consider an itemset's exact global support.¹¹ However, the correctness of DARM algorithms depends on each itemset's global support, without which we can't find a rule's exact confidence. So, if rules are generated using a partial support of itemsets, discrepancies will arise in the resultant rule set.

To maintain an association rule's correctness and compactness, ODAM sticks with the first approach. However, our technique significantly reduces the overall message exchange costs. How does it do this?

Most parallel and distributed ARM algorithms are based on sequential Apriori, because of its success in sequential setting.¹² Hence, directly adapting an Apriori algorithm won't significantly improve performance over frequent itemsets generation or overall DARM performance. To perform better than Apriori algorithms, we must focus on their problems.

The performance of Apriori ARM algorithms degrades for various reasons. It requires n number of database scans to generate a frequent n -itemset. Furthermore, it doesn't recognize transactions in the data set with identical itemsets if that data set is not loaded into the main memory. Therefore, it unnecessarily occupies resources for repeatedly generating itemsets from such identical transactions. For example, if a data set has 10 identical transactions, the Apriori algorithm not only enumerates the same candidate itemsets 10 times but also updates the support counts for those candidate itemsets 10 times for each iteration. Moreover, directly loading a raw data set into the main memory won't find a significant number of identical transactions because each transaction of a raw data set contains both frequent and infrequent items.

To overcome these problems, we don't generate candidate support counts from the raw data set after the first pass. This is because itemsets that are infrequent in the first pass cannot generate frequent itemsets in a subsequent pass. To efficiently generate candidate support counts of later passes, ODAM eliminates all infrequent items after the first pass and places those new transactions into the main memory. This technique not only reduces the average transaction length but also reduces the data set size significantly, so we can accumulate more transactions in the main memory. The number of items in the data set might be large, but only a few will satisfy the support threshold.¹⁵ Moreover, the number of infrequent itemsets increases proportionally for higher support thresholds.

Nevertheless, when we remove infrequent 1-itemsets from each transaction, the chances of finding similar transactions increases. Consider the sample data set in Figure 1a. If we load the data set into the main memory, then we only find one identical transaction (ABCD), as Figure 1b shows. However, if we load the data set into the main memory after eliminating infrequent items from every transaction (that is, items that don't have 50 percent of support and thus don't occur once in every second transaction—in this case, itemset E), we find more identical transactions (see Figure 1c). This technique not only reduces average transaction size but also finds more identical transactions.

Transactions	
No.	Items
1	ABCD
2	BC
3	AB
4	ABCDE
5	ACD
6	ABE
7	CDE
8	AD
9	BCE
10	ABCD

(a)

Transactions	
No.	Items
1,10	ABCD
2	BC
3	AB
4	ABCDE
5	ACD
6	ABE
7	CDE
8	AD
9	BCE

(b)

Transactions	
No.	Items
1,4,10	ABCD
2,9	BC
3,6	AB
5	ACD
7	CDE
8	AD

(c)

Figure 1. A (a) data set with (b) identical transactions and (c) transactions after pruning infrequent items.

One performance improvement FDM seeks is to generate fewer candidate itemsets. So, any distributed algorithms based on Apriori can easily achieve these performance improvements by incorporating these techniques. For example, consider three sites. After the first iteration, the global frequent 1-itemset F_1 is $\{A, B, C, D, E, F, G\}$. $A, B,$ and C are locally large and heavy at site S_1 , $B, C,$ and D are locally large and heavy at site S_2 , and $E, F,$ and G are locally large and heavy at site S_3 . Subsequently, in the second pass, FDM generates candidate itemsets from those frequent 1-itemsets that are globally and locally frequent at each site. Site support counts of candidate itemsets at site S_1 are $\{AB, AC, BC\}$; S_2 is $\{BC, BD, CD\}$; and S_3 is $\{EF, EG, FG\}$. Therefore, we reduce the total number of candidate itemset generation. In fact, this technique reduces significantly the number of candidate itemsets when numbers of disjoint itemsets among different sites are high. Consequently, we adopted this technique for our O DAM algorithm.

THE ODAM ALGORITHM

We assume each ODAM site has the same tasks as sequential association mining, except it broadcasts support counts of candidate itemsets after every pass.

Figure 2 shows ODAM's pseudocode. ODAM first computes support counts of 1-itemsets from each site in the same manner as it does for the sequential Apriori. It then broadcasts those itemsets to other sites and discovers the global frequent 1-itemsets. Subsequently, each site generates candidate 2-itemsets and computes their support counts. At the same time, ODAM also eliminates all globally infrequent 1-itemsets from every transaction and inserts the new transaction (that is, a transaction without infrequent 1-itemset) into memory. While inserting the new transaction, it checks whether that transaction is already in the memory. If it is, ODAM increases that transaction's counter by one. Otherwise, it inserts the transaction with a count equal to one into the main memory. After generating support counts of candidate 2-itemsets at each site, ODAM generates the globally frequent 2-itemsets. It then iterates through the main memory (transactions without infrequent 1-itemsets) and generates the support counts of candidate itemsets of respective length. Next, it generates the globally frequent itemsets of that respective length by broadcasting the support counts of candidate itemsets after every pass.

```

NF = {Non-frequent global 1-itemset}
for all transaction  $t \in D$  {
    for all 2-subsets  $s$  of  $t$ 
        if ( $s \in C_2$ )  $s$  .sup ++ ;
     $t' = \text{delete\_nonfrequent\_items} ( t );$ 
    Table.add (  $t'$  );
}

send_to_receiver (  $C_2$  );
/*Global Frequent support counts from receiver*/

 $F_2 = \text{receive\_from\_receiver} ( F_g );$ 
 $C_3 = \{\text{Candidate itemset}\};$ 
 $T = \text{Table.getTrasactions} (); k = 3;$ 
while ( $C_k \neq \{\}$ ) {
    for all transaction  $t \in T$ 
        for all  $k$ -subsets  $s$  of  $t$ 
            if (  $s \in C_k$  )  $s$  .sup ++ ;
     $k ++ ;$ 
    send_to_receiver (  $C_k$  );
    /*Generating candidate Itemset of  $k + 1$  pass*/
     $C_{k + 1} = \{\text{Candidate itemset}\};$ 
}

```

Figure 2. Pseudocode for the Optimized Distributed Association Mining algorithm.

Because ODAM eliminates all globally infrequent 1-itemsets from every transaction and inserts them into the main memory, it reduces the transaction size (the number of items) and finds more identical transactions. This is because the data set initially contains both frequent and infrequent items. However, total transactions could exceed the main memory limit.

To deal with this problem, we propose a technique that fragments the data set into different horizontal partitions. Then, from each partition, ODAM removes infrequent items and inserts each transaction into the main memory. While inserting the transactions, it checks whether they are already in memory. If yes, it increases that transaction's counter by one. Otherwise, it inserts that transaction into the main memory with a count equal to one. Finally, it writes all main-memory entries for this partition into a temp file. This process continues for all other partitions. During the writing, a tag is placed in front of every transaction to specify how many times that transaction exists in a partition. After eliminating infrequent items from each partition, ODAM iterates through the new data set (that is, the temp file) and generates the globally frequent itemsets of various lengths.

Message exchange optimization

In the CD algorithm, each local site generates support counts and broadcasts them to all other sites to let each site calculate globally frequent itemsets for that pass.^{3,12} So, the total number of messages broadcast from each site equals $(n - 1 * |C|)$. We can calculate the total message size using

$$T_{messages} = \sum_{i=1}^n (n - 1) * C,$$

where n is the total number of sites and C is number of candidate itemsets.

In contrast with CD, ODAM sends support counts of candidate itemsets to a single site, which calculates the globally frequent itemsets for that pass. We refer to the sites that send local support counts as the *sender* and the site that generates the globally frequent itemsets is the *receiver*. For example, with three sites, two broadcast their local support counts of candidate itemsets to the third site. The third site is responsible for generating that iteration's globally frequent itemsets. The total number of messages broadcast from a sender site to a receiver site equals $(1 * |C|)$.

Once the receiver site generates globally frequent itemsets, it broadcasts them to all sender sites. The total number of messages broadcast from the receiver is $(n - 1 * |F_G|)$. We can calculate the total message broadcasting size (the aggregate of sender and receiver sites messages) using

$$T_{messages} = (n - 1) * C + (n - 1)F_G,$$

where n is the number of sites, C is the candidate itemsets, and F_g is the globally frequent itemsets.

Example 1. Consider three sites $S_1, S_2,$ and S_3 . After the first iteration, suppose the set of large 1-itemsets is $\{A, B, C\}$. In the next iteration, each site has candidate itemsets equal to $\{AB, AC,$ and $BC\}$. If we consider sites S_1 and S_2 as senders and site S_3 as the receiver, then after the second iteration, the support counts of S_1 is $\{AB, AC$ and $BC\}$ and S_2 is $\{AB, AC$ and $BC\}$. These will be sent to the receiver site S_3 , which will generate this iteration's globally frequent itemset. If this iteration's globally frequent itemsets are equal to $\{AB$ and $BC\}$, site S_3 will then broadcast them to all sender sites. If we calculate the total message size, we find that the ODAM algorithm only broadcasts 10 messages whereas CD broadcasts 18 messages.

Some might argue that this optimization could cause a bottleneck in the receiver site when we increase the total number of participating sites. We argue that the receiver site receives the same number of candidate itemsets from all sender sites that a particular site would receive in the CD algorithm. For example, for four sites, each with four candidate itemsets, each site in the CD algorithm will send its own four candidate itemsets to other sites and will receive 12 candidate itemsets ($3 \times 4 = 12$) from other sites. Our proposed technique's receiver site will receive the same number of candidate itemsets ($3 \times 4 = 12$) from all sender sites and will broadcast a maximum of four globally frequent itemsets to sender sites. Furthermore, in most situations (that is, when globally frequent itemsets are less than the candidate itemsets), the receiver site sends fewer frequent itemsets to the sender sites.

The total number of candidate itemsets becomes large when the number of globally frequent itemsets is large in the previous pass. For example, if there are 500 global frequent 1-itemsets, the total number of candidate itemsets will be 124,750. However, all of those candidate itemsets are not globally frequent. Based on this rationale, we can further reduce the message exchange size if each sender sites sends only its local frequent itemsets instead of the whole candidate itemsets to the receiver site.

Sending only the local frequent itemsets from all sender sites doesn't always let the receiver site generate globally frequent itemsets for that pass because each sender site's local frequent itemsets are not identical. So, to find the globally frequent itemsets, the receiver sites further requests to sender sites and receives their infrequent support counts. Hence, we calculate the total message size using

$$F_T = F_{L1} \cup F_{L2} \cup F_{Lj} \dots \dots \dots \cup F_{L/T}$$

$$F_D = F_T \cap F_{L1} + F_T \cap F_{L2} + F_{LT} \cap F_{Lj} + \dots \dots \dots + F_T \cap F_{L/T}$$

$$T_{messages} = \sum_1^{n-1} F_L + (n - 1) + F_G + 2 * |F_D|,$$

where F_T is the union of all local frequent itemsets, F_D is the total number of disjoint local frequent

itemsets, F_L is the local frequent itemsets, and F_G is the globally frequent itemsets.

However, F_D is not scalable with respect to the total number of sites. This is because the receiver site sends and receives many requests if F_D is large. To overcome this shortcoming, we propose a method that calculates the probability of each itemset of F_D . Because the receiver site has the support counts of local frequent itemsets of all sender sites for that pass and the previous pass, it uses a probability function (P) that lets it determine the probability of a particular itemset of F_D . Based on that probability, it sends requests to other sender sites. To illustrate more precisely, consider the next example.

Example 2. Again, consider three sites S_1 , S_2 , and S_3 . Each site assigns a data set such as D_1 , D_2 , and D_3 , and each data set has 100, 150, and 250 transactions, respectively. We consider the global support threshold is 10 percent—in other words, we are interested in only those itemsets that have globally occurred 50 times. After the first iteration, suppose the set of global 1-itemsets = $\{A, B, C\}$.

In the next iteration, each site has candidate itemsets equal to $\{AB, AC, \text{ and } BC\}$. If we consider site S_1 and S_2 as senders and site S_3 as the receiver, then after the second iteration, the frequent itemsets support counts of S_1 are $\{AB - 10 \text{ and } BC - 16\}$, and S_2 is $\{AC - 16 \text{ and } BC - 22\}$. These will be sent to the receiver site S_3 , which calculates this iteration's the global frequent itemset. The receiver site has a frequent itemsets support count of S_3 equals $\{AB - 24, AC - 23, \text{ and } BC - 10\}$.

Because support of itemset AB is not frequent at site S_2 and AC is not frequent at site S_1 , site S_3 only has partial support counts of itemsets AB and AC . S_3 requires the global support count of itemsets AB and AC to determine whether those itemsets are globally frequent. So, it sends a request to site S_2 for the local support count of AB and S_1 for local support count AC . However, when the receiver site uses the probability function to determine whether the probability of those itemsets is high or low, it doesn't send these requests immediately. The receiver site calculates the probabilities of AB and AC .

In this case, these probabilities are low because AB and AC need to occur 16 and 11 times in site S_2 and S_1 to obtain a high probability. However, if AB and AC occur that many times in each of those sites, those itemsets become locally frequent and should be included in the local frequent itemsets. In this case, S_3 would receive the support counts of AB and AC from S_2 and S_1 in the initial message broadcast. So, the receiver site doesn't send any requests, and it discovers that the itemset BC is the only global frequent itemset for that pass. So, when we calculate the total message size, the optimized technique only broadcasts six messages whereas CD broadcasts 18 messages.

Complexity analysis

Our proposed message optimization technique reduces the communication cost because it only requires the local frequent itemsets to generate the globally frequent itemsets for that pass. Furthermore, it measures the probability of each disjoint local frequent itemset F_D . So, it only broadcasts messages for those local frequent itemsets that have higher probability and ultimately turn out to be globally frequent itemsets.

The communication complexity of our proposed method is $O(|C_R + P(F_D)| * n)$, where C_R is the intersection of all local frequent itemsets, $P(F_D)$ is the total number of disjoint local frequent itemsets that have higher probability, and n is the total number of sites. $|C_R + P(F_D)|$ is scalable with n because, in general, C_R never increases if we increase the total number of sites. Additionally, $P(F_D)$ is a probability function that remains constant or might increase linearly (that is, dependent on the data set) if we increase the total number of sites.

In comparison, CD requires communication complexity, where C is the candidate itemsets for that pass and n is the total number of sites. FDM requires $O(|C_P| * n)$ communication complexity, where C_P is the union of all local frequent itemsets and n is the number of sites. However, C_P increases as the number of sites n increases, so each FDM polling site sends and receives more requests, making FDM not scalable.

PERFORMANCE EVALUATION

We have extensively studied ODAM's performance to confirm its effectiveness. We implemented ODAM using C++. We established a socket-based, client-server distributed environment to evaluate ODAM's message reduction techniques. Each site has a receiving and a sending unit and assigns a specific port to send and receive candidate support counts. Because the candidate itemsets that each site generates will be based on the global frequent itemset for the previous pass, the candidate itemsets are identical among various sites.

We chose two real data sets for this performance study. [Table 1](#) shows the characteristics of each data set, including the number of items, average transaction size, and number of transactions of each data set. Both data sets are taken from the University of California, Irvine, Machine Learning Data Set Repository.¹³ Many ARM algorithms^{5,7,16} use these data sets as a benchmark for performance study. The Connect-4 data set is very dense and can produce many long frequent itemsets, even for high support values.⁷ In contrast, Cover Type is relatively sparse and uses low support thresholds to generate frequent itemsets.

Table 1. Data set characteristics

Name	Average transaction size	Number of items	Number of records
Cover Type	55	120	581,012
Connect-4	43	130	67,557

We organize our performance evaluation experiments in two parts. First, we execute ODAM with CD in a single site to compare how much time each algorithm takes to generate n length of frequent itemsets. Then, we compare the total communication cost (that is, the number of messages exchanged of ODAM, FDM, and CD) between different sites to generate the globally frequent itemsets.

Frequent itemsets generation

To show how ODAM can efficiently generate support counts, we conduct an experiment in a single site with different support, comparing total execution time between ODAM and CD. Because the total number of sites is equal to one, CD will perform the same as the sequential Apriori algorithm. FDM is also based on sequential Apriori and will work exactly the same as CD when the number of sites equals one.

Figure 3 shows ODAM and CD's total execution time for generating the frequent itemsets of various lengths using the UCI data sets. Both algorithms require more time when generating longer candidate itemsets. Generating support counts of candidate itemsets for each iteration takes approximately three times longer than it takes for the previous iteration. This is identical for both algorithms. However, ODAM removes a significant number of infrequent 1-itemsets from every transaction after the first pass, so it finds a significant number of identical transactions. As Figure 3 shows, ODAM outperforms CD in all cases. After eliminating infrequent items, ODAM doesn't enumerate candidate itemsets multiple times for any identical transaction. Furthermore, it requires a minimal number of comparison and update operations to generate support because it doesn't require comparison and update operations multiple times for similar transactions. In contrast, CD takes longer because each transaction contains exactly 43 items, thus requiring numerous comparison and update operations to candidate itemsets' generate support counts.

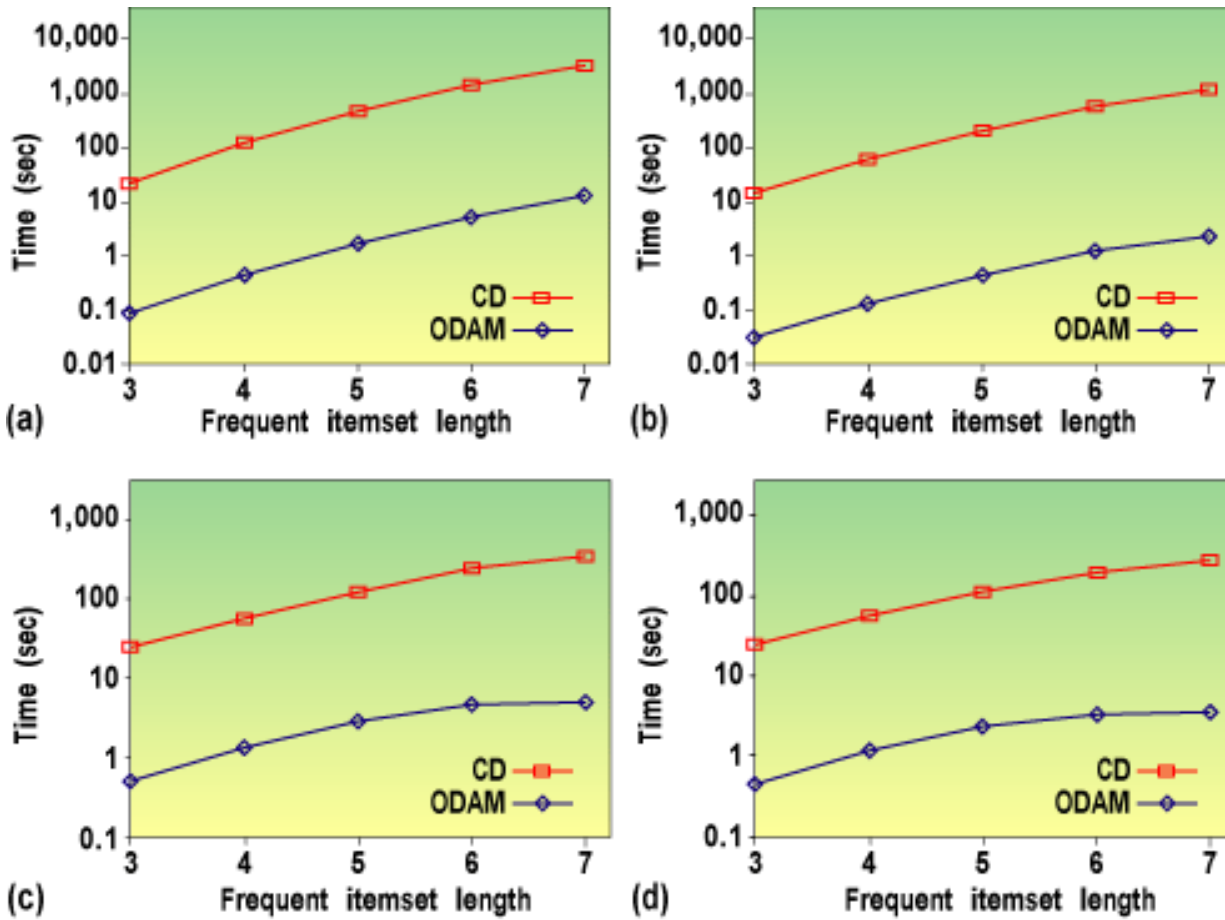


Figure 3. ODAM versus the Count Distribution algorithm in a single node with (a) Connect-4 data with 70 percent support, (b) Connect-4 data with 80 percent support, (c) Cover Type data with 0.5 support, and (d) Cover Type data with 1 percent support.

Message exchange

To compare the number of messages that ODAM, FDM, and CD exchange among various sites to generate the globally frequent itemsets in a distributed environment, we partition the original data set into five partitions. To reduce the dependency among different partitions, each one contains only 20 percent of the original data set's transactions. So, the number of identical transactions among different partitions is very low. Figure 4 depicts the total size of messages (that is, number of bytes) that ODAM, FDM, and CD transmit to generate the globally frequent itemsets with different support values.

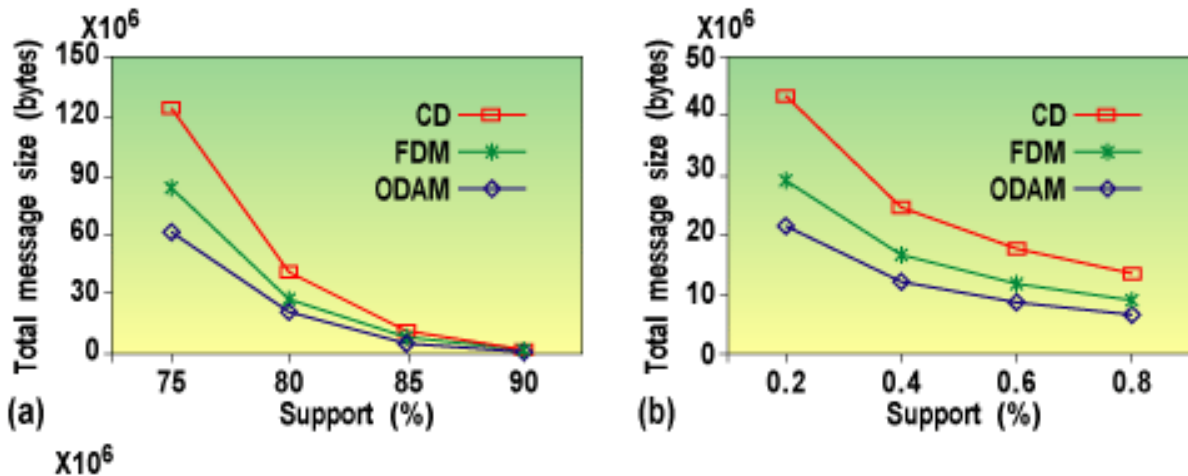


Figure 4. Total message size that ODAM, FDM, and CD transmit to generate the globally frequent itemsets for (a) Connect-4 data and (b) Cover Type data.

As Figure 4 shows, ODAM exchanges less messages among different sites to generate globally frequent itemsets. In all cases, ODAM reduces the communication cost by 60 to 80 percent compared to CD. In each site, CD exchanges messages with all other sites after every pass, and consequently the message exchange size increases when we increase the number of sites. ODAM reduces communication cost by 25 to 40 percent compared with FDM because FDM sends each support count to the polling site. After receiving a request, each polling site sends a polling request to all remote sites other than the originator site. Upon receiving the polling request from all other sites, the polling site computes whether that candidate itemset is globally frequent and broadcasts only globally frequent itemsets to all other sites. Hence, it exchanges more messages because each polling site sends and receives support counts from remote sites. It also needs to send global support counts to all participating sites when a candidate itemset is heavy and subsequently increases the communication cost. Furthermore, each polling site receives polling requests only from one site. Therefore, without receiving the support counts of remote sites, we can't presume whether an itemset is heavy.

One of distributed message optimization's main goals is scalability. To show whether our proposed message optimization techniques is scalable with respect to the number of sites, we conducted an experiment in which we increased the total number of sites from 10 to 100 (see Figure 5).

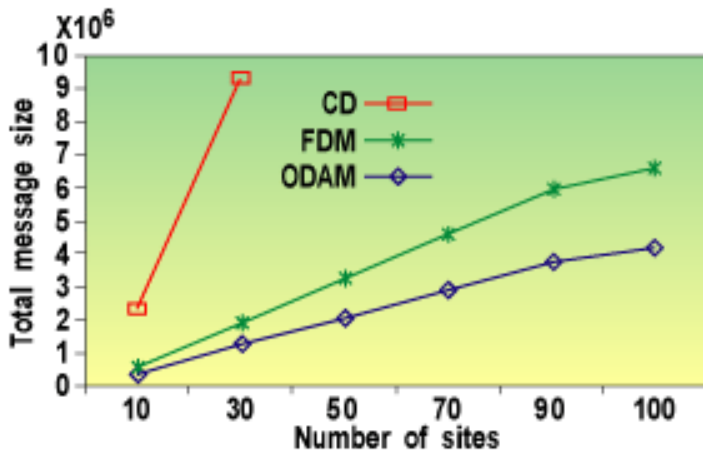


Figure 5. Scalability of ODAM versus CD and FDM (average transaction size = 43, $C_k = 6$, and support = 85%).

Our proposed optimization technique's message exchange size increases linearly as we increase the number of sites. It exchanges fewer messages than FDM, so ODAM is more scalable. Nevertheless, we expected this because ODAM generates fewer messages than FDM, especially when the number of participating sites was five, as in the previous experiment. On the other hand, CD broadcasts its candidate support counts to all other sites and subsequently receives support counts of others, broadcasting numerous messages when we increase number of sites.

CONCLUSION

ODAM provides an efficient method for generating association rules from different datasets, distributed among various sites. In future work, we plan to investigate how to efficiently perform DARM on different organizations in different domains. Because security and privacy is a common issue for data mining application, we'll also investigate how to maintain DARM's privacy without increasing overall communication costs.

References

1. R. Agrawal and R. Srikant , "Fast Algorithms for Mining Association Rules in Large Database,"*Proc. 20th Int'l Conf. Very Large Databases (VLDB 94)*, Morgan Kaufmann, 1994,pp. 407-419.
2. R. Agrawal and J.C. Shafer , "Parallel Mining of Association Rules,"*IEEE Tran. Knowledge and*

Data Eng. , vol. 8, no. 6, 1996,pp. 962-969;

<http://csdl.computer.org/comp/trans/tk/1996/06/k0962abs.htm>.

3. D.W. Cheung , et al., "A Fast Distributed Algorithm for Mining Association Rules," *Proc. Parallel and Distributed Information Systems*, IEEE CS Press, 1996,pp. 31-42;

<http://csdl.computer.org/comp/proceedings/pdis/1996/7475/00/74750031abs.htm>.

4. A. Savasere , E. Omiecinski, and S.B. Navathe , "An Efficient Algorithm for Mining Association Rules in Large Databases,"*Proc. 21st Int'l Conf. Very Large Databases (VLDB 94)*, Morgan Kaufmann, 1995, pp. 432-444.

5. J. Han , J. Pei, and Y. Yin , "Mining Frequent Patterns without Candidate Generation,"*Proc. ACM SIGMOD Int'l. Conf. Management of Data* , ACM Press, 2000,pp. 1-12.

6. M.J. Zaki and Y. Pin , "Introduction: Recent Developments in Parallel and Distributed Data Mining,"*J. Distributed and Parallel Databases* , vol. 11, no. 2, 2002,pp. 123-127.

7. M.J. Zaki , "Scalable Algorithms for Association Mining,"*IEEE Trans. Knowledge and Data Eng.* , vol. 12 no. 2, 2000,pp. 372-390; <http://csdl.computer.org/comp/trans/tk/2000/03/k0372abs.htm>.

8. J.S. Park , M. Chen, and P.S. Yu , "An Effective Hash Based Algorithm for Mining Association Rules,"*Proc. 1995 ACM SIGMOD Int'l Conf. Management of Data* , ACM Press, 1995, pp. 175-186.

9. M.J. Zaki , et al., *Parallel Data Mining for Association Rules on Shared-Memory Multiprocessors* , tech. report TR 618, Computer Science Dept., Univ. of Rochester, 1996.

10. D.W. Cheung , et al., "Efficient Mining of Association Rules in Distributed Databases,"*IEEE Trans. Knowledge and Data Eng.*, vol. 8, no. 6, 1996,pp. 911-922;

<http://csdl.computer.org/comp/trans/tk/1996/06/k0911abs.htm>.

11. A. Schuster and R. Wolff , "Communication-Efficient Distributed Mining of Association Rules," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, 2001,pp. 473-484.

12. M.J. Zaki , "Parallel and Distributed Association Mining: A Survey,"*IEEE Concurrency* , Oct.-Dec. 1999,pp. 14-25; <http://csdl.computer.org/comp/mags/pd/1999/04/p4014abs.htm>.

13. C.L. Blake and C.J. Merz , UCI Repository of Machine Learning Databases, Dept. of Information and Computer Science, University of California, Irvine, 1998; www.ics.uci.edu/~mllearn/MLRepository.html.

14. T. Shintani and M. Kitsuregawa , "Hash-Based Parallel Algorithms for Mining Association Rules,"*Proc. Conf. Parallel and Distributed Information Systems*, IEEE CS Press, 1996, pp. 19-30; <http://csdl.computer.org/comp/proceedings/pdis/1996/7475/00/74750019abs.htm>.

15. C.C. Aggarwal and P.S. Yu , "A New Approach to Online Generation of Association Rules,"*IEEE Trans. Knowledge and Data Eng.* , vol. 13, no. 4, 2001,pp. 527-540; <http://csdl.computer.org/comp/trans/tk/2001/04/k0527abs.htm>.

16. G.W. Webb , "Efficient Search for Association Rules,"*Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD 00)*, ACM Press, 2000,pp. 99-107.

Mafruz Zaman Ashrafi is a PhD student at Monash University. His research interests include data mining, distributed computing, e-commerce, and security. He received his MS in computer science from RMIT University. Contact him at School of Business Systems, Monash Univ., Clayton 3800, Australia; mafruz.ashrafi@infotech.monash.edu.au.

David Taniar is a senior lecturer at Monash University. His research interests include parallel

databases, data mining, parallel and distributed computing, and e-commerce. He received his PhD from Victoria University. He is editor in chief of the *Int'l J. Data Warehousing and Mining* and is a fellow of the Royal Society of Arts, Manufacturers and Commerce and of the Institute for Management Information Systems. Contact him at School of Business Systems, Monash University, Clayton 3800, Australia; david.tania@infotech.monash.edu.au.

Kate Smith is an associate professor at Monash University. Her research interests include data mining, neural networks, operations research, channel assignment in cellular phone networks, genome analysis using data mining techniques, and Web mining. She received her PhD in electrical engineering from the University of Melbourne. Contact her at School of Business Systems, Monash University, Clayton 3800, Australia; kate.smith@infotech.monash.edu.au.

Related Links

Purchase or Digital Library members log in:

- Design and Evaluation of a Parallel HOP Clustering Algorithm for Cosmological Simulation
(<http://csdl.computer.org/comp/proceedings/ipdps/2003/1926/00/19260082aabs.htm>), *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)* (Apr. 2003)

- Microarray Gene Expression Data Association Rules Mining Based On JG-Tree
(<http://csdl.computer.org/comp/proceedings/dexa/2003/1993/00/19930027abs.htm>), *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA 03)* (Sept. 2003)

- Data Mining for Very Busy People
(<http://csdl.computer.org/comp/mags/co/2003/11/ry022abs.htm>), *Computer* (Nov. 2003)