A DIC-based Distributed Algorithm for Frequent Itemset Generation

Preeti Paranjape-Voditel,
Department of Computer Applications, Shri Ramdeobaba Kamla Nehru Engineering College, Nagpur, Maharashtra, India
Email: preetivoditel@gmail.com
Dr.Umesh Deshpande
Department of Computer Science,Visvesvaraya National Institute of Technology (VNIT),Nagpur, Maharashtra, India
Email: uad@vnitnagpur.ac.in

***Abstract*— A distributed algorithm based on Dynamic Itemset Counting (DIC) for generation of frequent itemsets is presented by us. DIC represents a paradigm shift from *Apriori*-based algorithms in the number of passes of the database hence reducing the total time taken to obtain the frequent itemsets. We exploit the advantage of Dynamic Itemset Counting in our algorithm- that of starting the counting of an itemset as early as possible at the different site as soon as they become frequent at atleast one site. Hence, our algorithm shows remarkable improvement in the amount of time taken because of reduction in the number of passes of the database and comparatively lesser number of candidates generated. Distributed frequent itemset counting and association rule generation have basically used algorithms based on *Apriori* or *Sampling*. This is the first algorithm which is based on DIC.**

***Index Terms*— Distributed Association Rule Mining, dynamic Itemset Counting (DIC), Optimistic Messaging DIC**

I. INTRODUCTION

ARM has been used extensively for the classical problem of market basket analysis where it is required to find the buying habits of customers. Determining what products customers are likely to buy together can be very useful for planning and marketing. Association rules are used to show the relationships between these data items.

Many centralized algorithms exist for Association Rule Mining(ARM) [9], [10], [14], [15], [11]. Most of the algorithms depend on the discovery of frequent itemsets for generation of association rules. Since the total number of itemsets is exponential in terms of the number of items, it is not possible to count the frequencies of these sets by reading the database in just one pass.

Different algorithms for the discovery of association rules aim at reducing the number of passes by generating candidate sets, which are likely to be frequent itemsets. They attempt to eliminate infrequent sets as early as possible. Dynamic Itemset Counting (DIC) [1] is one such algorithm, which does not wait for a complete database pass to start counting the candidate itemsets. It therefore reduces the number of passes of the database and generates fewer number of candidate itemsets.

Why distributed ARM? With the presence of multinational companies at different geographical locations across the globe, the data they need for decision making is inherently distributed. It is necessary to analyse the data to allow company-wide activities such as planning, marketing and sales. Analyzing data locally is not enough. A straightforward solution is to transfer all data to a central site where data mining is done. However even when such a site is available, it may incur huge communication costs to transfer the local datasets because of their sizes. Sometimes the local data cannot be transferred because of the security or privacy of the datasets. Distributed Association Rule Mining (DARM) is an active field in which global association rules are formed for the distributed data. The performance affecting issues in a distributed environment are the disk I/O minimization, the time required for synchronization between the nodes and the message transmission over the network.

Almost all distributed ARM methods have been based on two sequential algorithmic paradigms:*Apriori* [3] and *Sampling* [9].We have designed our algorithm Optimistic messaging DIC (OPT-DIC) on the Dynamic Itemset Counting (DIC)algorithm. OPT-DIC focusses on disk I/O minimization by reducing the number of database passes and has almost no issues of synchronization between the nodes. It generates far fewer candidate sets than *Apriori*-based, level-wise algorithms because the nodes start counting an itemset early and only if it is frequent at atleast one node. This also reduces to a very large extent the number of bytes transmitted over the network.Our algorithm does not send the data but the counts of itemsets over the network thus security and privacy of the datasets is preserved.

The rest of the paper is organized as follows. Section II discusses the existing centralized algorithms and DIC. Section III deals with the issues in Distributed Association Rule Mining and the work done in the field of Distributed Association Rule Mining. Section IV discusses the Optimistic Messaging DIC algorithm.Section V discusses the

This paper is based on "An Optimistic Messaging Distributed Algorithm for Association Rule Mining ," by Preeti Paranjape-Voditel, Umesh Deshpande, which appeared in the Proceedings of IEEE Indicon-2009, Ahmedabad, Gujarat,India, December 2009.

doi:10.4304/jsw.6.2.306-313

results. We conclude with Section VI.

II. Association Rule Mining

Let *I*=$i_1, i_2, ...i_m$ 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*. The support of an itemset *X* is the number of transactions in which the itemset occurs as a subset. An itemset is frequent or large if its support is more than some user defined minimum support threshold δ. Thus support is the number of transactions in the database that contain the itemset *X*. An association rule is an implication of the form $X \Rightarrow Y$ where $X \subset I, Y \subset I$ and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds in the transaction set *D* with *confidence c* if c% of transactions in *D* that contain *X* also contain *Y*. The rule $X \Rightarrow Y$ has *support s* in the transaction set *D* if *s%* of transactions in *D* contain $X \cup Y$. The problem of mining association rules is to generate all association rules that have a certain user-defined minimum support and confidence.

Several centralized algorithms exist for Association Rule Mining. One of the first algorithms is *Apriori*, [2], [3] on which most of the parallel algorithms are based. Apriori is an iterative, level-wise algorithm which uses a bottom-up search starting with the counting of frequent 1- itemsets. It generates these itemsets after a complete scan of the database. It then uses a self-join to find the 2-itemsets from the frequent 1-itemsets. It then scans the database to find the frequent 2-itemsets and continues this process till the maximal itemsets are generated. The number of passes is equal to the size of the maximal n-itemset. It uses the large itemset property that is any subset of a large itemset must be large. The large itemsets are also said to be *downward closed* because if an itemset satisfies the minimum support requirements so will its subsets. Hence, if we know that an itemset is small then we need not generate its supersets as candidates because they will also be small.The performance of *Apriori* directly depends on the length of the longest frequent itemset. A remarkable breakthrough in sequential algorithms was achieved by the Dynamic Itemset Counting (DIC) [1] algorithm which represents a shift in the method in which frequent itemsets are generated. Since Dynamic Itemset Counting (DIC) forms the basis of our distributed algorithm, we would discuss this algorithm in detail.

A. Dynamic Itemset Counting (DIC)

Dynamic Itemset Counting (DIC) [1] is an algorithm which reduces the number of passes made over the data while keeping the number of itemsets which are counted in any pass relatively low. In the first *M* transactions the algorithm starts counting the 1-itemsets. After *M* transactions for a given minimum support threshold, if any of the itemsets exceeds the minimum support threshold in those *M* transactions, then we start counting the 2-itemsets before waiting for a complete scan of the database. In this way, DIC starts counting the 1-itemsets and then quickly adds counters for the 2,3,4,...k-itemsets. We will define this *M* as a checkpoint. DIC uses these checkpoints *M* transactions apart. DIC counts the frequent itemsets and the minimal small itemsets. Minimal small itemsets are those itemsets which form the boundary between the frequent itemsets and the infrequent ones. Their subsets are frequent itemsets. For every itemset, the counting stops from the same point from where it started i.e after one complete database pass. Thus an itemset can be considered for counting at the next checkpoint instead of waiting until the end of the previous pass.

If the data is fairly homogeneous and for small values of *M*, DIC takes very few passes. If the data is non-homogeneous or it is very correlated, it may not be realized that an itemset is actually large until it has been counted in most of the database.This effect can be reduced considerably with randomizing the order of the transactions. The most important issue in the performance of any ARM algorithm is the type of data structure used to keep track of the many itemsets generated. Particularly the data structure should support the addition of new itemsets, the incrementation of counters of those itemsets and maintaining the itemset states as those that are being counted or active and those which have been counted over the entire database. When itemsets become large the counting of the supersets should be started. The incrementation of the counters has to be done efficiently otherwise the performance of the entire algorithm may degrade.

The data structure used in DIC is a trie in which each itemset is sorted by its items.Every itemset that has to be counted or has been counted has a node associated with it as do all of its prefixes. The empty itemset is the root node and every itemset is attached to the root node. All itemsets are attached to their prefixes containing all but their last item. Every node stores the last item in the itemset it represents, a counter, as to where in the file its counting was started, its state and its branches if it is an interior node. The branches point to the supersets of the itemsets. These operations are performed at every checkpoint.

III. Distributed Association Rule Mining

In centralized data mining the main concern for the efficiency of a data mining algorithm is its I/O and CPU time. The I/O time is the number of diskreads or the number of passes of the database made by the algorithm.In a distributed environment the communication cost is added which is determined by the network bandwidth and the number of messages that are sent across the network. Count Distribution, Data Distribution, Candidate Distribution [4], ODAM(An Optimized Distributed Association Rule Mining Algorithm [5], [6], [7] are a few of the modified versions of *Apriori*. [8] is a distributed algorithm based on *Sampling*.

The Count Distribution (CD) algorithm focuses on minimizing communication. In the first pass, each processor dynamically generates its local candidate set depending

on the items actually present in its local data partition. Hence candidates counted by different processors may not be identical. Each processor exchanges local counts to develop global candidate counts. C_k is a set of candidate *k-itemsets* or potentially frequent itemsets where $1 \leq$ k $\leq N$ and *N* is the number of nodes in the distributed network . After each scan each processor broadcasts C_k and synchronization takes place at this step. L_k, a set of frequent *k- itemsets* or itemsets with minimum support is now generated from C_k. Each processor then decides to terminate or continue to the next pass and as all processors have the same L_k, this decision will be identical. Thus every processor scans its local data asynchronously in parallel and synchronizes at the end of each pass to develop global counts.

In the Data Distribution algorithm, each processor counts mutually exclusive candidates. Thus as the number of processors is increased, a large number of candidates can be counted in one pass. On a *n*-processor configuration, Data Distribution will be able to count in a single pass a candidate set that would require *n* passes in CD. The first pass of the algorithm is the same as CD where all the candidate 1-itemsets are counted. For all passes greater than 1, processor P^i, where $1 \leq$ i $\leq N$ generates C_k from $L_{(k-1)}$. P^i retains only $1/N^{th}$ of the itemsets, that it will count. If candidate set C_k generated by P^i is C_k^i then all such C_k^i sets are disjoint and their union is the original C_k. Processors exchange L_k^i so that every processor has the complete L_k for generating the candidate itemset C_{k+1} for the $(k+1)^{th}$ pass. The drawback of this algorithm is that every processor must broadcast its local data to all other processors in every pass.

The Candidate Distribution algorithm partitions the data and candidates in such a way that each processor may proceed independently. In some pass *l*, where *l* can be determined heuristically, the frequent itemsets L_{l-1} are divided between processors in such a way that a processor generates a unique candidate set irrespective of the other processors. Till pass *l*, Candidate Distribution behaves similar to Count or Data Distribution. At the same time data is repartitioned so that a processor counts its candidate set independently. No communication of counts or data tuples is done except for pruning the local candidate set. This information is sent asynchronously and processors do not wait for complete information to arrive. Each processor opportunistically starts counting the candidate sets using whatever information has arrived. Experimentation involving Count Distribution, Data Distribution and Candidate Distribution [4] has shown that Count Distribution (CD) outperforms the other two. We have compared our algorithm against CD.

IV. Optimistic Messaging DIC

We present a distributed algorithm based on DIC namely Optimistic Messaging DIC (OPT-DIC). OPT-DIC runs DIC at each node.

DIC reads *M* transactions and performs all operations of incrementation of the counters and adding supersets of items which become frequent.

We call every *M* (which may vary for each node, depending on the size of the database), in OPT-DIC, a checkpoint.

In this algorithm we also send and recieve messages at this checkpoint. At each node at every checkpoint, messages in the incoming queue are checked for counts of itemsets which have become potentially frequent at other sites. If counting for those itemsets has not begun at that node, it begins counting for that itemset.

With this step we would like to mention that OPT-DIC starts counting only those itemsets which have become locally frequent at atleast one node. This reduces the number of candidates who may ultimately not contribute to the frequent itemset generation.

The node sends messages at checkpoints regarding the counts of itemsets which can be potentially frequent without waiting for complete counting of that itemset. Because of this particular aspect of the algorithm, we describe the algorithm as an optimistic messaging distributed algorithm. This initiates early counting of that itemset at other sites. The main advantage of this algorithm is that it does not wait to synchronize with the other sites. It reports the potential candidate itemsets as soon as they turn potentially frequent at the next checkpoint. This leads to a significant reduction in the number of passes of the database as compared to CD.

A. Description of OPT-DIC

In OPT-DIC every site needs to maintain certain information with respect to every other site. To maintain this information, the messages used in OPT-DIC are:

1. T_i - number of transactions present at node N^i
2. $C^i_{candidate}$ - candidate itemset at node N^i
3. C^i_{inter} - candidate itemset with its intermediate local count at node N^i,
4. C^i_{final} - candidate itemset with its final local count at node N^i,
5. F_i - Final message at node N^i, indicating completion of counting.

Initially each node broadcasts message T_i, sending its number of transactions to every node. DIC is run locally at every node. Each node initiates counting of an itemset *I* at the *(n-1)* other nodes, if that itemset looks potentially frequent at its site. It does so by broadcasting a message $C^i_{candidate}$ at the next checkpoint. After recieving the counts, it can locally decide whether *I* is globally frequent or not. Each node N^i maintains information associated with each itemset *I* to indicate whether it is locally frequent or infrequent and globally frequent or infrequent or unknown. The messages to be broadcast are maintained in a message queue Q_{out} and broadcast at the next checkpoint. All incoming messages are kept in an incoming message queue Q_{in}. When a node N^i counts the itemset *I* over the entire database and if *I* is locally or globally frequent, a message C^i_{final} with the count of *I* is generated at N^i. If no itemsets are to be counted locally,

N^i broadcasts F_i and waits for incoming messages. If *(n-1)* F_j, where $(1 \leq j \leq n; i \neq j)$ messages are recieved by N^i, then all nodes have completed counting.

B. Algorithm Optimistic Messaging DIC (OPT-DIC)

```
For each node N^i, we perform the following
begin
  Run DIC locally
  At each checkpoint at N^i
     I. For any message m in Q_in do
     begin
     1. If m is C^j_final
           update count of I for j
     2. If m is C^j_inter
           a) Start counting of I,
           if not yet started
           b) If I is globally frequent
           OR globally infrequent
           OR UNKNOWN
                 Mark I
     end

     II. If counting of I complete do
     begin
     If ( I is locally infrequent
     AND globally frequent)
     OR
     (I is locally frequent
     AND globally infrequent)
            Add C^i_final for I in Q_out
     end

     III. During counting of I do
     begin
     If I becomes a local candidate
     OR I turns locally frequent
     AND I marked UNKNOWN
            Add C^i_inter OR C^i_candidate_l
            to Q_out
     end

     IV.If no itemsets are to be counted
           Broadcast F^i.
     N^i waits for incoming messages
     If (n-1) F^j messages received
     (1 ≤ j ≤ n; i ≠ j)
        Counting at all nodes is complete.

end
```

If messages are sent at every checkpoint and if there are *l* such checkpoints and if we are grouping messages for every checkpoint, then *(n-1)* such messages will be sent at every checkpoint by each site, if there are *n* such sites.The number of these checkpoints will also depend on the number of database scans i.e if we have *l* such checkpoints in one database scan and an average of *p* such scans and if C_i^j are the average number of candidate sets generated at the i^{th} checkpoint by site *j* then the message complexity in the worst case will be:
At the i^{th} checkpoint the number of messages are:
$(n-1) \times C_i^j$
Total messages in one pass:
$(n-1) \times \sum_{i=1}^{l} C_i^j$

It is not necessary that at every checkpoint a message is generated.We consider the worst case here, where we consider a message at every checkpoint and average candidate sets as C_i^j.
If *p* (where *p* can be a non-integer) is the number of database scans, the number of messages broadcast by each node in the worst case are:
$p \times (n-1) \times \sum_{i=1}^{l} C_i^j$

C. The Algorithm

The basic essence of DIC lies in the fact that counting for itemsets starts very early. At every checkpoint, all itemsets which have become locally frequent are marked so. These have become frequent recently, they are not marked frequent by any other site (indicated by a message from any other site initiating counting for that itemset). Such itemsets are broadcast to the other sites to initiate counting at the other sites. Thus counting is initiated at all sites only if an itemset becomes frequent at atleast one site.
The OPT-DIC algorithm retains the basic essence of DIC in the distributed version in terms of communicating the itemsets at the next checkpoint when they become likely candidates locally. This helps the other sites in starting the counting for those itemsets, if counting for them has not already been started. This is done at the next possible checkpoint. In the worst case the communication complexity is such that there is a message broadcast at every checkpoint.So if *D* is the size of the database, *M* is the interval which represents the number of transactions between two checkpoints, D/M will represent the number of checkpoints.If *N* is the number of sites and *p* is the number of passes the message complexity in the worst case will be $D*(N-1)*p/M$. To reduce the number of messages the interval between checkpoints (the value of *M*) can be increased but many a times this adversely affects the performance as there is a delay in conveying the candidate itemsets which are locally frequent.
The disadvantage of CD and many *Apriori*-based algorithms is that the number of bytes transmitted increase rapidly with the number of nodes.This is also because a lot of globally infrequent but locally frequent candidate itemsets are broadcast between nodes. This factor is reduced to a great extent in OPT-DIC as a node starts counting an itemset only if it is frequent at atleast one site. Without communicating with the other nodes CD proceeds with its entire database pass and then broadcasts the itemsets generated. This may contain many itemsets which may not contribute towards the global frequent itemset generation.

V. Experimentation and Analysis

We have compared OPT-DIC and CD using a Discrete Event-based Simulator. We have tested Optimistic Messaging DIC and CD on the benchmark datasets namely

mushroom dataset [12], the retail dataset and two synthetic datasets T10I4D100K and T40I10D100K generated from [13]. The mushroom dataset is a multivariate,dense dataset with 8124 transactions, 119 items and an average transaction size of 23.The retail dataset is sparse with 16,470 items, 88,162 transactions and average transaction size of 10.The datasets T10I4D100K and T40I10D100K are sparse with 1000 items and 1,00,000 transactions each and the average transaction size as 10 and 40 respectively. The performance metrics we have considered are the total time taken, number of passes, number of diskreads, number of bytes and the number of messages. The number of diskreads take into account the number of passes so we have not elaborated on the number of passes.

From the above datasets, for T10I4D100K, we have experimented on the value of minimum support δ as 1% and M as 100. For T40I10D100K, we have taken δ as 4% and M as 100. For the Retail dataset, δ has been taken as 10% and M as 100. Since Mushroom is a dense dataset, we have taken a higher value of δ as 50% and M as 100. We have packetized the mesages generated at the end of each pass in CD and at a checkpoint in OPT-DIC with a MTU of 1500 bytes and a header of 20 bytes. To calculate the disk access time we have considered the seek time (3 msec), disk latency time (2 msec) and disk transfer rate (1000 Mbps). To calculate time for transmission across the network, latency time (15 msec) and bandwidth (1 Mbps) have been considered. We now discuss the performance of the algorithms according to the above performance metrics on equipartitions and on gaussian paritions.

A. Results for Equipartitions

We have tested the results of the algorithm by equipartitions and on gaussian partitioning, i.e. to check the results on variable paritions as well as for similar partitions.

(1) Total time taken : Due to the huge size of the database and the disk and network latency, time taken is heavily dependent upon the maximum number of passes and the number of messages transmitted. In case of CD, sites synchronize at the end of every pass. This makes the sites with minimum records wait for the sites with maximum records to send their counts.In case of OPT-DIC, none of the sites try to synchronize. They send messages but as long as there is data to be mined locally, they do not wait for messages from other sites.

From Figure 1 we observe that the time required for OPT-DIC for T10I4D100K is around 52% less than that required for CD and this reduction in time is almost constant for an increase in the number of nodes. The time required for OPT-DIC for T40I10D100K is around 32% less than that required for CD and this reduction in time is almost constant for an increase in the number of nodes. We have observed that the rate of reduction in time required for OPT-DIC as compared to CD for Retail is almost constant between 60 to 70%. We observe that the rate of

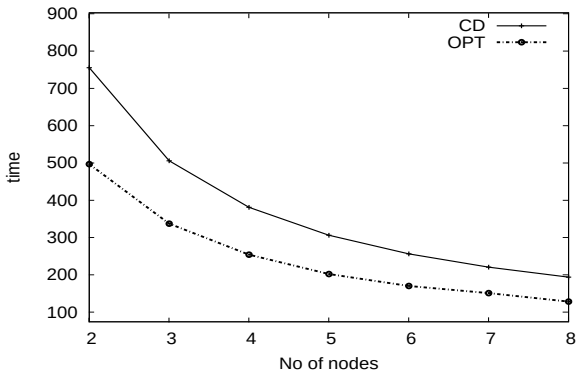

Figure 1. Time taken for T10I4D100K for $\delta = 1\%$ and M=100

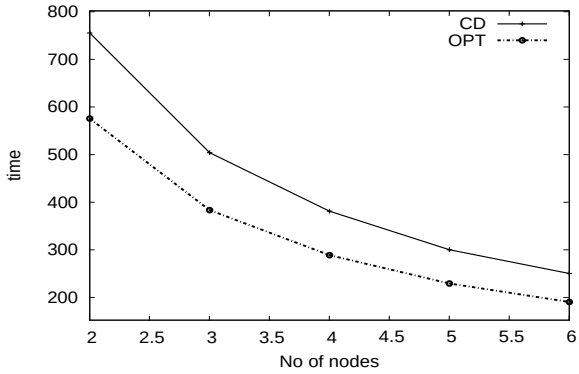

Figure 2. Time taken for T404D100K for $\delta = 4\%$ and M=100

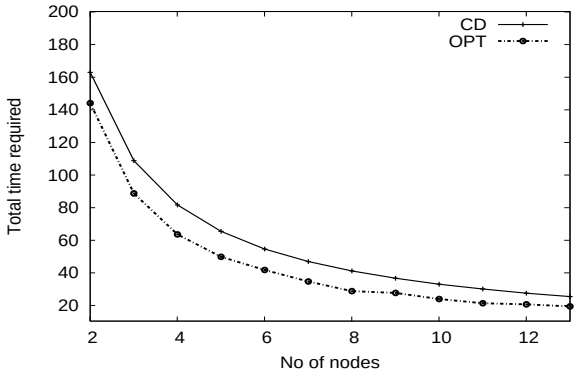

Figure 3. Time taken for mushroom for $\delta = 50\%$ and M=100

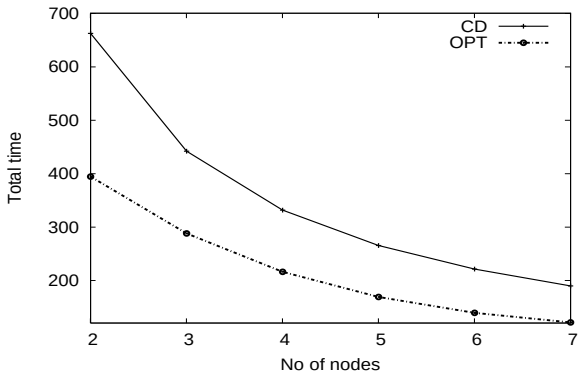

Figure 4. Time taken for retail for $\delta = 10\%$ and M=100

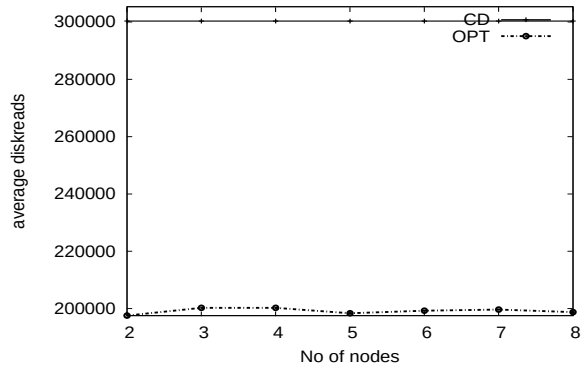

Figure 5. Diskreads required for T10I4D100K for $\delta = 1\%$ and M=100

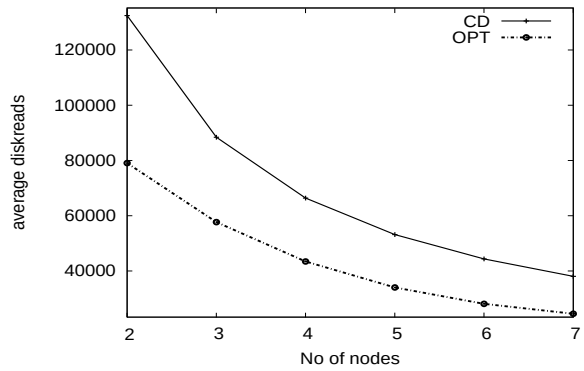

Figure 6. Diskreads required for 'retail' for $\delta = 10\%$ and M=100

reduction in time required for OPT-DIC as compared to CD for Mushroom increases with the increase in number of nodes.

(2) Number of diskreads : The number of passes in OPT-DIC are much less than those in CD as seen in figures 5 and 6. Local counts of all itemsets is always maintained. As a result if *n* globally frequent itemsets are not locally frequent at the same site, they would never be used to generate a candidate at the next level. Not only does this reduce the number of candidates but it can save upto one extra pass. All these factors reduce the number of diskreads.
We observe that the diskreads in OPT-DIC are around 50% less than those in CD. This is a major component of the time required.

(3) Number of Bytes transmitted:
The number of bytes transmitted by CD in the first pass is quite high as it sends the counts and names of all the frequent 1-itemsets. In subsequent passes, only counts of all candidates are transmitted.Please refer to figures 7 and 8. In the case of OPT-DIC the count and name of each itemset with cardinality more than 1 is transmitted. But the number of candidate itemsets generated in CD are far more than those in OPT-DIC.

(4) Number of messages transmitted: By messages here, we mean the number of packets transmitted over the network. In case of OPT-DIC, if some itemsets have

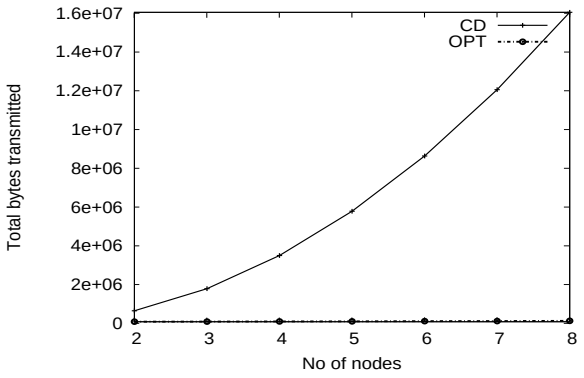

Figure 7. Bytes generated for T10I4D100K for $\delta = 1\%$ and M=100

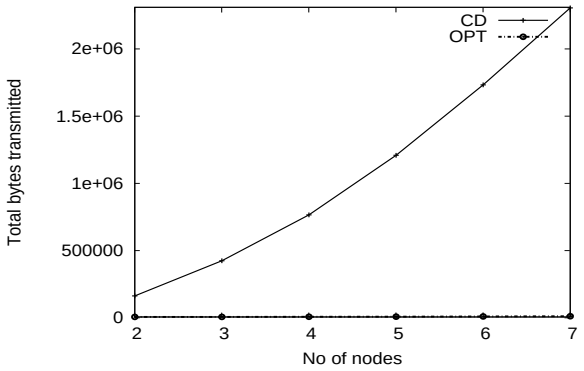

Figure 8. Bytes generated for retail for $\delta = 10\%$ and M=100

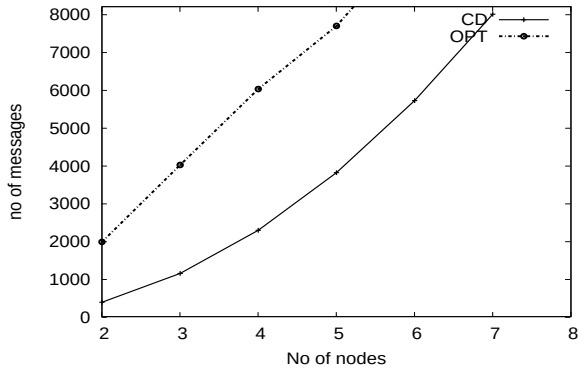

Figure 9. Messages generated for T10I4D100K for $\delta = 1\%$ and M=100

turned potentially frequent, messages are generated at that checkpoint.In the worst case, a message is broadcast at every checkpoint. Hence the number of messages is higher in OPT-DIC compared to CD. But though the number of messages is higher, OPT-DIC fares much better than CD in the time taken.

B. Results for Gaussian Partitioning

We have partitioned the database using the gaussian distribution which represents the actual distribution of data at various sites in an actual distributed setup. Using similar performance metrics as applied to equipartitioning, we

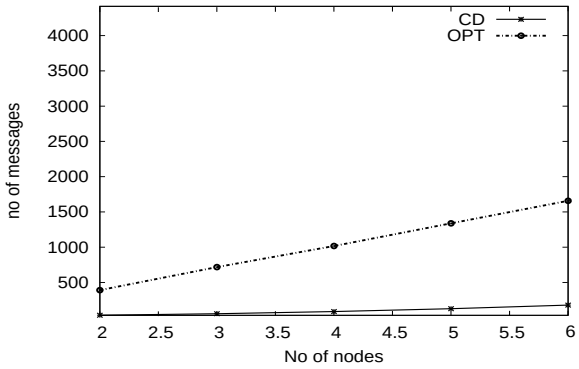

Figure 10. Messages generated for mushroom for $\delta = 50\%$ and *M*=100

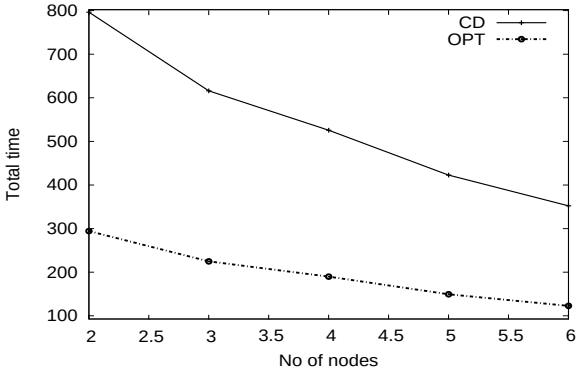

Figure 11. Time taken for gaussian distribution of T10I4D100K $\delta = 1\%$ and *M*=100

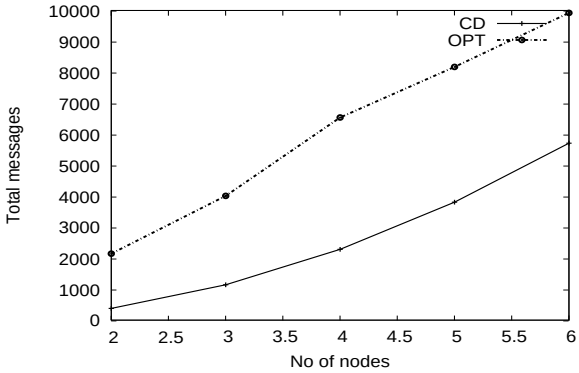

Figure 12. Messages generated for gaussian distribution of T10I4D100K for $\delta = 1\%$ and *M*=100

analyse the results for gaussian partitions.

(1) Time taken: We observe that with gaussian partitioning the performance gain in terms of reduction in the time required in OPT-DIC is much higher than that with equipartitioning.
(2) Bytes transmitted:
The total bytes transmitted in gaussian partitioning show much reduction than in equipartitioning
(4) Messages transmitted: There is definitely a reduction in the number of messages in gaussian distribution for both CD as well as OPT-DIC but the messages in OPT-DIC are more than those in CD.

The above results show that OPT-DIC outperforms CD with respect to the time taken, the maximum number of diskreads and the total number of bytes transmitted. It transmits larger number of messages than CD but since the total time required by OPT-DIC is much lower the other parameters offset the impact of larger number of messages.

VI. Conclusion

In this paper, we have presented an algorithm, OPT-DIC, based on Dynamic Itemset Counting which represents a different approach to frequent itemset generation in a distributed ARM environment. We have observed that compared to CD, OPT-DIC shows much higher performance gain on sparse as well as dense datasets.

Acknowledgment

We are thankful to Shreyas Belle and Rahul Suresh for their valuable contribution in the implementation of OPT-DIC.

References

[1] S.Brin and R.Motwani and J.Ullman and Shalom Tsur, *Dynamic Itemset Counting and Implication Rules for Market Basket Data*,SIGMOD Record, volume 6,number 2,pages 255-264,June 1997.
[2] R.Agrawal and T.Imilienski and A.Swami,*Mining Association Rules between Sets of items in large Databases*,Proc. of the ACM SIGMOD int'l Conf. on Management of Data,May,1993,pages 207-216.
[3] R.Agrawal and R.Srikant,*Fast Algorithms for Mining Association Rules*, Proceedings of the 20th VLDB Conference, Santiago, Chile,1993,
[4] R.Agrawal and J.Schafer,*Parallel Mining of Association Rules*,IEEE Transactions on Knowledge and Data Engineering",volume 8,number 6, pages 962-969,1996.
[5] M.Ashrafi and D.Taniar and K.Smith,*ODAM:An Optimised Distributed Association Rule Mining Algorithm*,IEEE Distributed Systems Online,5,3, March,2004,1541-4922.
[6] Cheung.D and Han.J and Ng.V and Fu.A and Fu.Y,*A Fast Distributed Algorithm for Mining Association Rules*, Proc of Tnt'l Conf on Parallel and Distributed Information Systems, Miami Beach Florida, 31-44.
[7] Assaf Schuster and Ran Wolff,*Communication-Efficient Distributed Mining of Association Rules*,ACM SIGMOD, Boston,4,May,2001.
[8] Assaf Schuster and Ran Wolff and Dan Trock, *A high Performance Distributed Algorithm for Mining Association Rules*
[9] Toivonen.H, *Sampling Large Databases for association Rules*, VLDB Journal,pages = 134-145.
[10] Lin.D.I and Kedem.Z.M,*Pincer Search: A new algorithm for discovering the maximum frequent set*,Extending Database Technology,October-December, 1999,14-25
[11] Mohammed Zaki,*Parallel and Distributed Association Mining: A Survey*, IEEE Concurrency,October-December,1999,14-25.
[12] UCI Machine Learning Repository, *http://archive.ics.uci.edu/beta/datasets/Mushroom*.
[13] Srikant.R,*Synthetic Data Generation Code for association and sequential Patterns*,IBM Quest website at http://www.almaden.ibm.com/cs/quest.
[14] Margaret Dunham,*Data Mining, Introductory and Advanced Topics*, Pearson Education,2006.

[15] J.Han and M.Kamber,Data Mining, Concepts and Techniques,Morgan Kaufmann Elsevier Science India.

Preeti Paranjape-Voditel is currently a Ph.D. candidate at The Department of Computer Science and Engineering,Visvesvarayya National Institute of Technology (VNIT),Nagpur, Maharashtra, India. She received her MTech in Computer Science and Information Technology from the Indian Institute of Technlogy, Kharagpur, WestBengal, India and BE in Electronics from Walchand College of Engineering, Sangli, Maharashtra, India. She is presently working as an Assistant Professor in the Department of Computer Applications, Shri Ramdeobaba Kamla Nehru Engineering College, Nagpur, Maharashtra, India. Her research interests include Distributed Data Mining, Algorithms and Databases.

Umesh Deshpande received his PhD in Computer Science and Engineering in 2005 from the Indian Institute of Technlogy, Kharagpur, WestBengal, India. He recieved his Masters from the Indian Institute of Technlogy, Bombay, Maharashtra, India and BE from Visvesvarayya National Institute of Technology (VNIT),Nagpur, Maharashtra, India. He is currently an Associate Professor in the Department of Computer Science and Engineering at Visvesvaraya National Institute of Technology (VNIT),Nagpur, Maharashtra, India. His current research interests include distributed systems,real-time operating systems, multi-agent systems and Data Mining.