Parallel Genetic Algorithm Model to Extract Association Rules

Ahmad M. Taleb, Anwar A. Yahya
College of Computer Science and Information Systems
Najran University
Najran, Saudi Arabia
amtaleb@nu.edu.sa
anwaralthari@gmail.com

Nasser M. Taleb
College of Business Administration
Al Ain University of Science and Technology
Al Ain, UAE
nasser.taleb@aau.ac.ae

Abstract— **Over the past generation, the process of discovering interesting association rules in data mining and knowledge discovery has become a cornerstone of contemporary decision support environments. While most of the existing algorithms do indeed focus on discovering high interestingness and accuracy relationships between items in the databases, they tend to have limited scalability and performance. In this paper, we discuss a Parallel Genetic Algorithm Model (PGAM) that has been designed as a scalable and high performance association rules engine. Experimental results demonstrate that the model offers the potential to optimize both scalability and performance in association rules mining.**

Keywords - Data mining; FP-Growth; Multi-objective evolutionary algorithms; scalability; performance; Parallel GA.

I. INTRODUCTION

Data mining and knowledge discovery in databases have been popular targets for researchers over the past 15-20 years, with papers published on a wide variety of related topics. One of the most important tasks in the data mining domain is the association rule mining that aims to find the relationships between the items that frequently appear in the databases' transactions, and additionally, to extract rules of the form IF Condition THEN Predication. The IF clause is called the rule condition that checks if the values of some attributes are true and the THEN clause is called the rule prediction that predicts a value for some goal attribute. In general terms, an association rule is a relation between attributes of the form if X then Y, Where $X \cap Y = \Phi$. It is known that the association rule mining is NP-hard problem because the search space is exponential with the number of itemset.

In 1993, the association rule problem was first introduced by Agrawal et al. [1]. They developed the Apriori algorithm which is the most famous algorithm to solve the association rule problem [2]. This algorithm is based on the support (frequency of the rule in the transactions) and confidence (truth of the rule in the transactions) of the rule. Most of the existing association rules algorithms built upon Apriori-based algorithm, as the Apriori algorithm was well understood and very famous. On the positive side, the improvements of the Apriori algorithm were very impressive in terms of measuring the quality of the generated rules by using a coherent set of multiple measures such as interestingness, comprehensibility, confidence, etc. Unfortunately, such algorithms still make the problem more complex and often provided limited scalability as they are ill-suited to handle massive databases with huge number of attributes and a lot of distinct values for each attribute.

Other approaches are based on Frequent Pattern Growth (FP-Growth) Algorithm [13] to extract association rules. The FP-growth is used to extract the frequent itmesets from the databases in two steps as follows: 1) Build a compact data structure called FP-tree using two passes over the databases and 2) Extract frequent itemsets from the FP-tree by the traversal through the FP tree. After detecting the frequent itemsets, then we can use a user defined parameter called confidence to generate the appropriate association rules. While the FP-Growth algorithm [13] needs only two passes over the datasets, a large amount of memory space is needed because the algorithm generate conditional FP-trees recursively.

For this reason, several parallel algorithms have been proposed in the literature to handle the association rule problem in massive data stores [18]. Scalability on these parallel algorithms was/is indeed acceptable as the partition of large databases and transactions often handles massive data stores. Of course, everything comes at a price and, in the case of parallel algorithms; runtime performance remains a big concern. Specifically, such algorithms often provided poor runtime performance due to the high synchronization and communication overhead and disk I/O cost.

The current paper discusses a parallel genetic-based algorithm to discover association rules called PGAM. The motivation of our design is to provide a high runtime performance and scalable engine for the association rule mining problem. Physically, the architecture is constructed as a federation of largely independent sibling servers, each responsible for a segment of the original transactions. Locally, each server stores, and processes its transactions to extract the association rules using the genetic algorithm that is very well suited to perform global search with less time complexity compared to other algorithms used in data mining problems. A parallel service layer transparently provides global merging and communication services as required. Specifically, the Parallel Genetic Algorithm Model (PGAM) described in this paper is capable of efficiently solving the association rule problems. Experimental

 ISBN: 978-1-61208-247-9

evaluation demonstrates that the combination of a shared nothing architecture and heavily optimized local genetic algorithm processing may indeed provide the cost effective scalability/performance pairing that is missing in existing association rule algorithms.

The paper is organized as follows. In Section 2, we briefly review recent work in the area. Basic preliminary material is then outlined in Section 3. We present the details of the parallel model in Section 4, including the genetic algorithm that ties the model together. In Section 5, we discuss the shared nothing architecture and show how the local servers are integrated into a single logical system. Experimental results are then provided in Section 6, with final conclusions in Section 7.

II. RELATED WORK

Association rule mining (ARM) is an important core data mining technique to discover patterns/rules among items in a large database of variable-length transactions. The goal of ARM is to identify groups of items that most often occur together. It is widely used in market-basket transaction data analysis, graph mining applications like substructure discovery in chemical compounds, pattern finding in web browsing, word occurrence analysis in text documents, and so on. Contemporary association rule mining (ARM) research began with the definition introduced by Agrawal et al. [1], an important data mining technique to discover rules among items in massive databases of large number of transactions. Moreover, Agrawal et al. developed the Apriori algorithm to solve the association rule mining problem. This algorithm focuses on the frequent itemsets generation sub-problem and subsequently the generation of the rules with minimum confidence. Subsequently, a number of researchers presented algorithms that are improvements to Apriori algorithm [10], [21]. FP-growth is another famous technique to extract the frequent itemset using minimum support and confidence [13]. More recent work in this area has tended to focus on the quality of the generated rule by considering more measures (mutli-objective algorithms) [10]. In general, scalability and performance were not addressed in the well known Apriori and FP-growth algorithms and their improvements.

Apart from the Apriori and FP-growth algorithms, a significant number of publications focused on generating the association rules using genetic algorithm [5]. Genetic algorithm was first developed by John Holland in 1975. It is based on the idea of survival of the fittest and the greedy approach and performs very well global search with less time. The GA works as follows:

1. An initial population is created. A Population is a group of individuals (Chromosomes) and represents a candidate solution. A Chromosome is a string of genes.
2. Select chromosomes with higher fitness.
3. Crossover between the selected chromosomes to produce new offspring with better higher fitness
4. Mutate the new chromosomes if needed.
5. Terminate when an optimum solution is found.

Ghosh et al. [11] proposed an algorithm to extract frequent itemsets using genetic algorithms. Dou [7] also developed an algorithm to find the maximal frequent itemsets using GA and some defined parameters such as individual identity, individual fitness, upgrade index and upgrade genes that are used in GA. The authors in [15] developed an algorithm for extracting the association rules using GA and without the specification of the user-defined minimum support and confidence. Finally, Hong [14] developed a two-phases GA algorithms to extract the association rules.

With respect to parallel algorithm for the ARM, a number of algorithms were developed based on the parallelization of the Apriori and FP-growth algorithms [3], [12], [16], [19]. Each attempted to effectively exploit the parallel hardware and architecture. Especially, the set of transactions (Databases) is partitioned into a number of subgroups and attempts to utilize the resources of the parallel system efficiently. In other words, each partition is assigned to an independent processor that makes the decision to process and terminate the algorithm. A few other parallel algorithms should be mentioned here. The Hori-Vertical algorithm [18] is a parallel algorithm where no node will be idle because a lot of new independent tasks that can be taken to process. The author in [18] proposed a new database partitioning that is based on dividing the database vertically and horizontally into equivalent parts. Eclat[20] makes vertical database partitioning and is another parallel algorithm to solve the ARM. Limine et al. [4] proposed the ”Workload Management Distributed Frequent itemsets mining” (WMDF) algorithm that is based on the horizontal database partitioning and it makes load balancing between system nodes. Parallelizations of the well-known sequential algorithms are discussed with many other parallel algorithms surveyed in [20].

III. PRELIMINARY MATERIALS

We can formally state the task of mining association rules over market basket as follows: let I={I1, I2, …, In} be the set of items/products and T={T1, T2, …, Tn} be the set of transactions in the database. Each of the transaction Ti has a unique ID and contains a subset of the items in I, called itemset. An association rule is an implication among itemsets of the form, $X \rightarrow Y$, where $X \cup Y \subseteq I$ and $X \cap Y = \emptyset$ [1][2]. An itemset can be a single item (e.g. mineral water) or a set of items (e.g. sugar, milk, red tea). The quality of the association rules can be measured by using two important basic measures, support(S) and confidence(C) [17]. Support(S) of an association rule is the percentage of transactions in the database that contain the itemset $X \cup Y$. Confidence (C) of an association rule is the percentage/fraction of the number of transactions that contain $X \cup Y$ to the total number of records that contain X. Confidence factor of $X \rightarrow Y$ can be defined as:

$$\textbf{Conf}(X \rightarrow Y) = \textbf{Support}(X \cup Y) / \textbf{Support}(X) \quad (1)$$

Most of the association rule algorithms generate the frequent itemsets –itemsets that are greater than a minimum support—and then generate association rules that have

 ISBN: 978-1-61208-247-9

confidence greater than minimum confidence. However, more metrics such as Comprehensibility and Interestingness can be used to have more interesting association rules [10]. Comprehensibility is measured by the number of attributes involved in IF part (condition) of the rule with respect to the THEN (prediction) part of the rule because the rule is more comprehensible if the conditions is less than the prediction. Comprehensibility of an association rule (X→Y) is measured as:

Comp (X→Y) = log(1+|Y|)+log(1+|XUY|) **(2)**

where |Y| and |XUY| are the number of attributes in the consequent side and the total rule, respectively.

Interestingness measures how much interesting the rule is [10]. The interestingness of an association rule (X→Y) is measured as:

Inter(X→Y) = [Support (XUY)/Support(X)]x [Support (XUY)/Support(Y)x[1 – Support(XUY)/|D|] **(3)**

Where |D| is the total number of records/transactions in the database. Using several measures, the association rule problem can be considered as a multi-objective problem.

Fig. 1(a) and (b) depict a small grocery sales database consisting of five products I = {M, R, S, T, W} and six transactions table that illustrates the purchase of items by customers.

Products/items	Abbreviation
Milk	M
Rice	R
Sugar	S
Tea	T
Water	W

(a)

Transaction	Items/products
1	MRTW
2	RSW
3	MRTW
4	MRSW
5	MRSTW
6	RST

(b)

Figure 1. (a) products/items database (b) Database transactions.

Fig. 2(a) shows all frequent itemsets containing at least three products and minimum support 50%. Fig. 2(b) illustrates sample association rules with four metrics (Support, Confidence, Comprehensibility and Interestingness).

In this paper, we choose to deal with the association rule mining as a multi-objective problem rather than one objective problem and to adopt the multi-objective evolutionary algorithm for mining association rules [8], [9] with emphasize on genetic algorithms. Genetic algorithm is an iterative procedure that is appropriate for situations such

Itemsets	Support
R	100%
W, RW	83%
M, S, T, MR, RS, RT, MRW	67%
MT, SW, TW, MRT, MTW, RSW, RTW, MRTW	50%

(a)

Association rules	Support	Confi-dence	Comprehen-sibility	Interest -ingness
M→R	67%	1	2.58	0.59
MR→T	50%	0.75	3	0.51
W→RS	50%	0.75	3	0.51
R→W	83%	0.83	2.58	0.71

(b)

Figure 2. (a) Frequent Itemsets with minimum support 50% (b) Sample association rules with three metrics

as large and complex search space and optimization problems. In order to use the genetic algorithm, the following points must be addressed [5]:

1. Encoding/decoding schemes of chromosomes (bit string, real-value string, etc.)
2. Population size: how many chromosomes are in population. Good population size is about 30-40.
3. Fitness value: the chromosomes must be ranked according to their fitness values (e.g. support, confidence, comprehensibility, etc. measures can be used to calculate the fitness value of an association rule).
4. Selection: select the chromosomes for next generation by using one of the selection scheme (Roulette wheel, Boltzman, Tournament , Rank, etc.).Note that it is important to use the elitism technique to make sure that the best chromosomes (association rules) that were generated at some intermediate generations will be kept as candidate solutions.
5. Crossover: single point crossover, two point crossover, multi-point crossover, uniform crossover, and arithmetic crossover.
6. Mutation: This to change the new chromosome (offspring) to prevent the algorithm from getting stuck. For binary encoding, the algorithm changes bits from 1 to 0 or vice versa.

IV. MINING ASSOCIATION RULES

The association rule engine described in this paper is a fully parallelized model. That being said, it is physically constructed as series of backend servers, each operates independently to extract the association rules from the database transactions that are housed in each of the nodes. This federated approach allows us to design and build a parallel association rule model by concentrating on the

optimization of the single node servers, rather than complex load redistribution policies. The end result is a parallel association rule engine that can directly exploit techniques developed for single node servers. In other words, the current engine can run on a single node, providing good performance that one would expect from a fully optimized association rules mining system. We therefore begin our discussion of the current model by looking at the partitioning of the transactions in the database and the execution model of the sibling (backend) servers. In Section 5, we will return to the question of how to efficiently integrate the individual nodes.

A. Transaction Database Partitioning

We start by looking at the partitioning of the transaction databases (e.g. market-basket problem) across all p backend nodes. Given that our aim is to balance the processing times for mining association rules across all p processors, a good partitioning mechanism is a necessity. We note that our focus in this paper is the database (set of transactions) of the market-basket problem with fix positions of products/items. The set of transactions are stored as bit strings where each string represents a transaction. Table 1 illustrates how the database (set of transactions) looks like.

Table I. Set of transactions (database of market-basket)

	Products/Items									
Transactions		A	B	C	D	E	F	G	H	…
	T1	0	1	1	0	1	0	0	0	…
	T2	1	1	1	0	1	0	0	1	…
	T3	1	0	0	0	1	0	1	0	…
	…									

The stripping technique is described below.

1. **Sort** the original transactions according to the number of items per transaction.

2. **Stripe** the transactions across all processors in a round robin fashion such that successive transactions are sent to the next processor in the sequence. For a network with *p* processors, a database of *n* transactions and *n* mod *p* != 0, a subset of processors receives one additional transaction.

The main goal behind this striping technique is that it dramatically increases the likelihood that the execution time required to extract the association rules will be proportionally distributed across the processors in the multi-computer architecture.

B. Multi-objective Genetic Algorithm

Recall that each backend node operates independently to extract the association rules from the transactions that are housed in each of the backend nodes. Specifically, each backend node executes independently an optimized multi-objective genetic algorithm to extract the association rules for its own transactions database. In our current work, we tried to solve the multi-objective association rule problem with the pareto based [22] genetic algorithm because it is always difficult to find out a single solution for multi-objective problem. Vilferdo Pareto suggested the non-dominance approach to solve multi-objective problems. His approach says " A solution, say a, is said to be dominated by another solution, say b, if and only if the solution b is better or equal with respect to all the corresponding objectives of the solution a, and b is strictly better in at least one objective". We start by discussing the characteristics of the pareto genetic algorithm used in to extract the multi-objective association rules.

The first task in the genetic algorithm is to define what the chromosomes represent (e.g. association rules, frequent itemsets and how (e.g. encoding/decoding). Since we decided to use the Pareto based genetic algorithm in ARM, then the chromosomes will be representing the possible association rules. Two famous approaches (Pittsburg or Michigan) can be used to encode the chromosomes [6]. Pittsburg is very suitable for classification rule mining, while Michigan is more suitable for association rule mining and encodes each part (antecedent and consequent) of the rule separately. More interestingly, Ghosh et al. [10] developed a better scheme for encoding/decoding the rules to/from binary chromosomes. According to Gosh [10], each item or product in the association rule is represented in two bits. If these two bits are 00 the product/item (No need to store the product/item value because the positions of values are fixed) value according to its position appears in the antecedent and if it is 11 then the value of the product appears in the consequent. The other two combinations, 01 and 10 indicate that the absence of the product's value in the rule. In our work, a chromosome represents a possible rule and is represented in binary format as follow: Given six products (ABCDEF), the rule AC → BE will look like 00 11 00 01 11 00. Note that we need k extra bits, where k is the number of items in the database. Note that chromosome data represents a possible association rule that consists whether or not a product/item exists in the association rule (no need to store the actual value of product/item because the positions of products are fixed)**.**

The fitness value for each chromosome is calculated by using a set of three complementary metrics, (1) Confidence (2) Comprehensibility and (3) Interestingness, to filter out the interesting rules. More specifically, an objective fitness function combines these metrics to calculate the fitness value of the chromosomes (possible rules) as the arithmetic weighted average confidence, comprehensibility and interestingness. The fitness function f(x) is defined as follow:

$$f(x) = (W1 * Confidence + W2 * Comprehensibility + W3 * Interestingness) / W1+W2+W3 \quad (4)$$

Where W1, W2, W3 are user defined weights each of the metric and W1+W2+W3 = 100. The weights of the metrics, used to calculate the fitness value, are defined by the user defined parameters (W1, W2 and W3). In other words, the user defined parameters are chosen according to the user's interestingness for each one of the metrics.

As mentioned in equations 1 and 3 that the support of the antecedent part, consequent part and the rule are essential in order to calculate the rule's metrics, as well as, the rule's fitness value. For this reason, we adopted the FP-tree structure [13] to compress a larger database and to avoid the

 ISBN: 978-1-61208-247-9

extensively scanning of the raw data set on disk multiple times, as might be done with a naive implementation. If two transactions share a common prefix, then the shared parts will be merged as long as the count is updated properly. There is a better chance that more prefix strings can be shared because the FP-Tree is created by ordering the items by their decreasing support. In our work, the FP-Tree is called SupTree that consists of set of nodes, each of which is defined as **N(value, counter, parentNode, childNode).** A node contains an item value, counter, pointer to the parent node and pointers to its children.

The FP-tree is constructed in two passes over the data-set. In the first pass, scan the data-set and find the support for each item and then sort the items in decreasing order based on their support. In the second pass, Algorithm 1 illustrates the construction of the in-memory tree (SupTree).

The FP-tree (SupTree) usually has a smaller size than the uncompressed data because typically many transactions share items and prefixes and it can fit in the main memory. Moreover, the order of the items by decreasing support minimizes the size of the FP-tree. However, if every transaction has a unique set of values, then the size of the tree is at least as the original database and even higher because of the need to store the pointers between the nodes and the counters.

Algorithm 1 (SupTree construction)

Input: Set of Transaction Table with fix positions of values
Output: FP-Tree Structure called SupTree
1: An array A of size n, where n is the total number of items in the dataset, is created to store the items in decreasing order with their support. E.g. A[0].item contains the item with the highest support (A[0].support).
2: Create the root of the SupTree and label it as null.
3: For each transaction in the database [t|T] wher the is first value and T is the remaining list,
    3.1: Call Insert_Tree ([t|T], SupTree)
4: **Function Insert_Tree([t|T], SupTree)**
    4.1: If SupTree.root has a child N and N.value = t.value Then
        4.1.1: increment the counter of N by 1,and update A**else**
        4.1.2: create a new node N(value, counter, parentNode,
            childNode) and do the following:
            N.Value = t.value, N.counter = 1;
            N.parentNode = SupTree.root.childNode;
            update array A (the parent node of N is linked to SupTree)
    4.2: If T is not empty
            4.2.1: Call Insert_Tree(T, N)

For example, Table 2 shows a sample data-set where 18 transactions and 6 items are exist. First, items in the transactions are sorted in decreasing order by their support. For our example, the order is **(f; a; c; b; d; e)** as shown in Table 3. Fig. 3 illustrates the FP-tree (SupTree) corresponding to the data of Table 2.

Table II. Sample data-set (18 transactions with 6 items)

TID	Items	TID	Items	TID	Items
T1	B, C, D, F	T7	B, D	T13	A, B,C, F
T2	A,D ,F, E	T8	A, C, F	T14	A, B, C, D, F
T3	A, B, C, F	T9	F	T15	A, B, C, D, E
T4	A, C	T10	E, F	T16	A,F
T5	B, F	T11	A, B, C, F	T17	A,D,F
T6	B, C, D	T12	C, F	T18	A,F

Table III. Support for each item

F	A	C	B	D	E
13	11	10	9	7	3

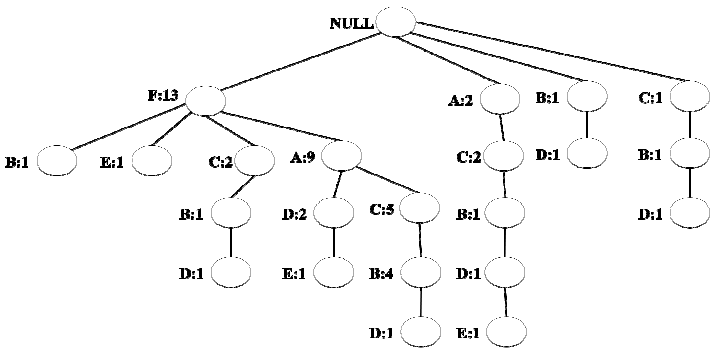

Figure 3. FP-Tree (SupTree) after reading the transactions

In our paper, the FP-tree and the in-memory array (A) will be used to calculate the support of any combinations of items/values without scanning the raw data set multiple times. Recall that each chromosome represents a possible rule and the position of values/items are fixed so that they are not mentioned within the chromosomes. Algorithm 2 shows how the FP-Tree (SupTree) and array (A) are used in very efficient way to calculate the support for any combination of items/values/attributes. In short, the combination of items will be ordered by using the same order of items in array A. We use a top-down process to find the support of the combination. The idea in Algorithm A is to eliminate all sub-trees that will not contribute in the support of the combination (search only sub-trees that may contain the combination of items). For example, given a combination ABC, the algorithm will search only sub-trees rooted at F and A first, then C then B and discard other sub-trees.

After representing the chromosomes and the fitness values, various genetic operators (selection, crossover, mutation, etc.) can be applied to them. Equations 1, 2, 3 and 4 with the FP-tree structure can be used in order to rank the chromosomes according to their fitness values. After

 ISBN: 978-1-61208-247-9

ranking the chromosomes, selection operator is used to select the best chromosomes to be in the next population. There are many selection techniques, but, in our paper the chromosomes are selected, for next generation, by the roulette wheel selection scheme. In case of ARM, we need to store the best rules found from the database. However, if we follow the standard genetic operators (selection, crossover, mutation) only, then the final population may not contain better rules that were generated at some intermediates generations. For this reason, we use the elitism technique to replicate the chromosomes ranked as 1 into the next population. If better chromosomes are generated by the standard genetic algorithm operations (selection, crossover and mutation), then replace the dominated chromosomes by the new generated one.

Algorithm 2 Support Calculation

Input: FP-Tree (SupTree) and array A and a list of items W
Output: Support of W
1: Order the items of W to be in the same order of items in array A.
2: Call function Find-Support(SupTree, W)
3: **Function Find-Support (SupTree, W)**
3.1: set Pos = A[W[0]].postion; Pos is the position of W[0] in Array A **(e.g. W[0] = A and Pos = 1)**
3.2 For i = 0 and i<= Pos do
For each child T in SupTree.children()
If (T.value() == A[i])
If(A[i] == W[0])
If(W.size() == 1)
then
return
T.counter;
Else W=W[1…n]
call function Find-Support(T, W)
endif
Else
Call function Find-Support (T, W)
Endif
endfor endfor

Due to the large number of items/products in the market-basket problem, thereby multi-point crossover operator is needed. In short, random positions (crossover points) in the strings (chromosomes) will be chosen and all bits before the first point will be copied from the first parent and all bits after that point and before the next point will be copied from the second parent, and so on until all crossover points are covered. After the crossover is performed, mutation takes place to prevent falling of all solutions in population into a local optimum of the problem. For binary encoding, the mutation procedure changes few randomly chosen bits from 1 to 0 and vice versa. The mutation usually occurs with a very low probability.

V. PARALLEL GENETIC ALGORITHM MODEL (PGAM)

The model in this paper has been designed as a parallel model to extract association rules. The data is partitioned and distributed to all nodes that are operated independently. The database partition is considered as a preprocessing step. In terms of the parallel model, it can be described at a high level as follows. The frontend node serves as an access point for all user defined parameters (mutation probabilities, number of association rules required). Parameters reception and management is performed at this point. The frontend distributes the required parameters to all backend nodes, collect final results from all backend servers, and prepare the final result as per the user requirements. In turn, the backend nodes are fully responsible for extracting the association rules form their local data. In addition, each node houses a *Parallel Service Interface* (PSI) component that allows it to identify its neighborhoods and when and how have to communicate with them. Fig. 4 illustrates the primary components, including the linkage between the sibling servers that are designed according to the ring topology.

With respect to the PSI, we choose to use the open source OpenMPI communication libraries because MPI minimizes the complexity of data transmission and communication within the parallel server. Therefore, utilizing the MPI libraries, the server can be constructed as a single MPI-based application. Specifically, the parallel server consists of a set of nodes (e.g. frontend and backend) that are executed simultaneously and subsequently communicate to each other. Standard precise and reliable operations (send, receive, gather, scatter, broadcast) can then be executed.

As mentioned above, the original set of transactions (datasets) is partitioned and distributed to each one of the backend nodes in round robin fashion. Once the original data (available in the frontend node) is distributed and received by the backend servers, the frontend node broadcasts the user parameters (number of association rules required, number of attributes in the antecedent, number of generations, etc.) and any pre-defined values (crossover and mutation probabilities, size of population, etc.) to all backend nodes. Because of the distribution technique of the original data and replication of the parameters on each of the backend nodes, our parallel mode is said to be load balanced parallel model for mining association rules. At this stage, we are ready to extract the association rules. For this, of course, we require the genetic algorithm along with the FP-Tree services described in Section 4. Algorithm 3 provides a high level description of mining association rules on the backend server instances. In short, identical parameters are sent to each node, where the local server uses these parameters to extract association rules on its local set of transactions. After the execution of all functions and before sending the local results to the frontend, a Parallel Fitness Calculation is performed across the parallel machine. The PSI provides this functionality. Specifically, each node P send the final population R to all other nodes (ring topology) in order to calculate the fitness values of the chromosomes with respect to all transactions found in the

 ISBN: 978-1-61208-247-9

original dataset. For example, if P is the current node and N backend nodes exist, then send R to (P+1)%N, (P+2)%N, … , (P+N-1)%N. At the end of this step, each node contains a local population with respect to the local data but the fitness values of the local chromosomes are according to all transactions found in all nodes. Finally, the local population results are returned to the frontend buffers where necessary processing takes place such as merging and ranking of all chromosomes and then return the appropriate association rules as per the user parameters.

following steps then remove the dominated chromosomes from this population.
9: Using the roulette wheel scheme along with the fitness values, select the chromosomes for next generation and replace the chromosomes of old population.
10: Perform multi-point crossover and mutation on these new chromosomes.
11: if the required number of generations is not completed, then go to Step 4

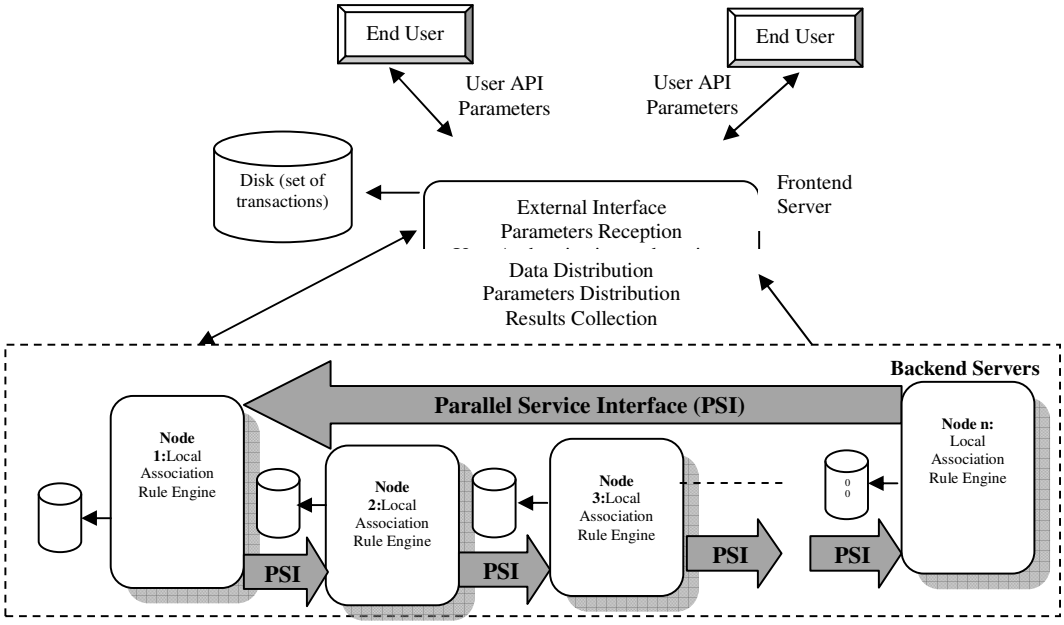

Figure 4: The core architecture of the parallel Association rule engine.

Algorithm 3 Backend Association Rules Engine

Input: A set of parameters received from the frontend and the local set of transactions.
Output: Population with a set of possible association rules sent to the frontend node.
1: Receive the user's parameters (uP) and any required pre-defined parameters from the front end (vP)
2: Load the local transaction and create the FP-Tree (SupTree) and array A by calling Algorithm 1.
3: Generate N chromosomes randomly; each chromosome represents a possible rule
4: Decode the chromosomes to get the values of the different items.
5: Using Algorithm 2, find the support of the antecedent side, consequent side and the rule.
6: Using equation 1, 2 and 3, find the confidence, comprehensibility and interestingness
7: Rank the chromosomes by calculating their fitness values (use equation 4).
8: Copy the chromosomes ranked as 1 into a separate population, if better chromosomes are generated from the

12: Do a Parallel Fitness Calculation by sending the final stored population (R) to all other backend nodes in the parallel model (using ring topology design as shown in Fig. 4).
13. Return result R to the frontend (collect R with MPI Allgather()).

VI. EXPERIMENTAL RESULTS

In terms of the environment, parallel evaluation was conducted on an 8-node (16 processors), Gigabit Ethernet Linux cluster, with each 2.6 GhZ ProLiant board housing 2 GB of memory. For the test database, we used the Synthetic transactional database generated by IBM Quest Market-Basket Data Generator to synthesize a transaction database. Specifically, we used 1000 unique items to create 10 Million records, each of which has average transaction length of 10. Default values of the genetic parameters are: Population Size = 40, crossover probability = 0.8, mutation probability = 0.02, the values of user-defined weights are chosen to be equal W1=0.33, W2=0.33 and W3=0.33 [11]. In the future,

 ISBN: 978-1-61208-247-9

the default values along with the user defined parameters will be changed to evaluate their affects to our algorithm.

We begin by looking at the performance of the proposed parallel genetic algorithm to extract association rules described in this paper. We have directly compared our model with some of the previous parallel association rules algorithms (Elcat, WMDF, HorVertical) using different number of sibling servers (1, 2, 4 and 8 nodes) in the system and minimum support of 0.5%, if needed. Note that the current server has only 8 nodes but our algorithm is designed to work on any parallel server regardless the number of nodes. Fig. 5 shows that the performance of our Parallel Genetic Algorithm Model (PGAM) to extract Association rules does indeed outperform previous parallel association rules algorithms. With respect to our algorithm, the running time consider the time to build and maintain the FP-Tree, communication time, receive results in the front end node and prepare the final list of association rules. This result is due to many reasons. First, our model is based on the Evolutionary Generic Algorithm that is very suitable to such problem (Association rules). Second, finding the support is going to be very fast by adopting the concept of FP-Tree. Third, the encoding/decoding schemes of chromosomes allow us to find the association rules directly without the idea of frequent itemsets. Fourth, the data portioning ensures the load balanced and that all nodes are contributing equally in extracting the association rules. Finally, our approach scans the database only once while the Eclat algorithm scans the database three times and the WMDF algorithm scans the database a lot of Times. Note that HoriVertical scans the database only once but it is not based on evolutionary algorithm.

In production environments, it is quite likely that association rule algorithms will be accessing databases (set of transactions) that are larger than the ones that can be conveniently tested in academic settings. As a result, it is important to provide some understanding of performance as set of transactions (databases) grow. Our scalability assessment begins with a look at performance patterns as the number of transactions increases from 10 million to 40 million records. In this experiment, we use 8 nodes to extract the association rules as the number of transactions vary. Fig. 6 shows the execution time as a function of number of transactions (database size). As can be seen in the figure, the running time is increased by a factor of 1.3 as the number of records in the database increases by a factor of two. The result is expected because the size of the FP-Tree would be almost the same as the number of transactions increases. Consequently, our Parallel Genetic Algorithm Model to extract association rules is very scalable in that an increase in the number of transactions is associated with nearly the same execution time. Note that other algorithms (Elcat, WMDF, HorVertical) focus on the parallel runtime performance to extract association rules therefore we did not compare our algorithm in terms of scalability with their algorithms.

Due to the current capacity of the nodes' memories and processors, we could not make experiments with larger data sets. But in theory the algorithm is designed to support large and big real-world data sets. In the future, we will upgrade the capacity of memory and processor in each one of the nodes to perform experiments with larger datasets.

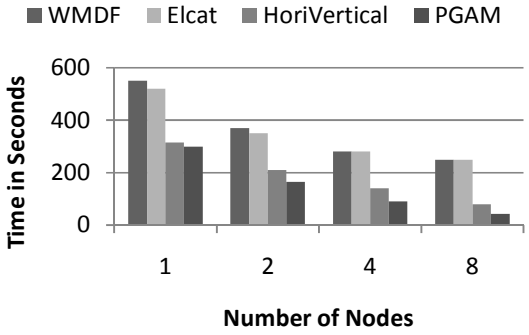

Figure 5. performance of our model (PGAM) versus other parallel algorithms

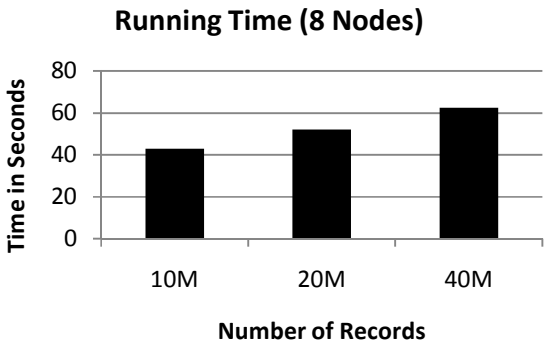

Figure 6. Running time as a function of the number of records

The parallel speedup graph illustrated in Fig. 7 depicts a speedup of approximately 7 (about 88% of optimal). The difference between observed speedup and optimal speedup is due to the Parallel Fitness Calculation used to calculate the fitness values of all chromosomes found in the parallel nodes.

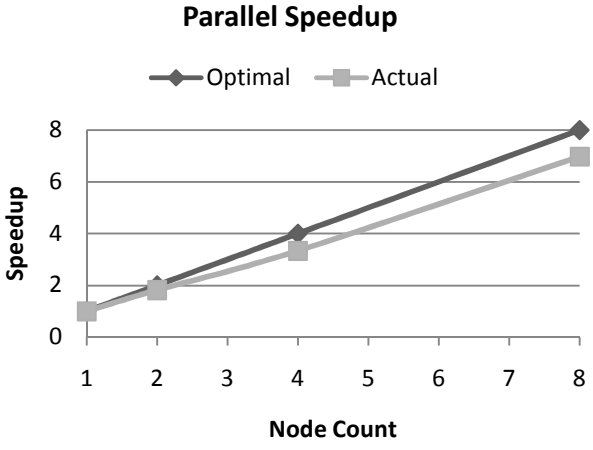

Figure 7. Parallel Speedup

VII. CONCLUSION

A great deal of association rules and frequent itemsets research has been published over the past 15-20 years. For the most part, however, researchers tend to focus on Apriori and FP-Growth algorithms and data structures for single node servers. Given the size of the underlying market-basket databases, coupled with the availability of modestly priced hardware and the advantages of genetic algorithm -- it was proved to perform global search with less time complexity and also very well suited for NP-hard problem such as ARM--, there exists great opportunity for the exploitation of cluster-based data mining servers by using GA. In this paper, we have discussed a Parallel Genetic Algorithm Model (PGAM) for association rules. Constructed as a federation of heavily optimized sibling servers, the current model demonstrates the potential to provide both high performance and scalability. Experimental evaluation in both multi-node scenarios suggests that the current model does indeed have the potential to achieve this objective.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," in Proc. of ACM SIGMOID Conf., pp. 207-216, 1993.

[2] R. Agrawal and R. Srikant, "Fast Algorithm for Mining Association Rules," in Proc. of the 20th VLDB Conf., pp. 487-499, 1994.

[3] R. Agrawal and J. Shafer, "Parallel Mining of Association Rules," In IEEE Transactions on Knowledge and Data Engineering, vol. 8, no. 6, pp. 962-969, 1996.

[4] L. Aouad, N. Le-Khac, and T. Kechadi, "Distributed frequent itemsets mining in heterogeneous platforms," Engineering, Computer and Architecture, Volume 1 (2007).

[5] S. Das and B. Saha, "Data Quality Mining using Genetic Algorithm," International Journal of Computer Science and Security, (IJCSS) Volume 3, Issue 2, pp. 105-112.

[6] S. Dehuri, A. Jagadev, A. Ghosh, and R. Mall, "Multi-objective Genetic Algorithm for Association Rule Mining Using a Homogeneous Dedicated Cluster of Workstations," American Journal of Applied Sciences 3 (11): 2086-2095, 2006 ISSN 1546-9239, 2006.

[7] W. Dou, J. Hu, K. Hirasawa, and G. Wu, "Quick Response Data Mining Model Using Genetic Algorithm," SICE Annual Conference, 2008, pp. 1214-1219.

[8] M. Fonesca and J. Fleming, "Multi-objective Optimization and Multiple Constraint Handling with Evolutionary Algorithms," Part I: A Unified Formulation. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans, 28(1), pp. 26-37, 1998.

[9] A. Freitas, "Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery," Advances in evolutionary computing: theory and applications, pp 819 – 845, 2003.

[10] A. Ghosh and B. Nath, "Multi–objective rule mining using genetic algorithms," Information Sciences 163, pp 123-133, 2004.

[11] S. Ghosh., S. Biswas., D. Sarkar., and P. Sarkar, "Mining Frequent Itemsets Using Genetic Algorithm," International Journal of Artificial Intelligence & Applications (IJAIA), Vol.1, No.4, October 2010

[12] E. Han, G. Karypis, and V. Kumar, "Scalable Parallel Data Mining for Association Rules," In Proceedings of the ACM SIGMOD International Conference on Management of Data, 1997, pp. 277-288.

[13] J. Han, J. Pei, and Y. Yin, "Mining Frequent patterns without candidate generation," 2000, In Proc. Of ACM-SIGMOD Int. Conf. on Management of Data, pp. 1- 12.

[14] T. Hong, J. Huang., W. Lin, and M. Chiang, "GA-Based Item Partition for Data Mining," 2011 IEEE, pp. 2238-2242.

[15] A. Islam, T. Chung**, "**An Improved Frequent Pattern Tree Based Association Rule Mining Technique," Information Science and Applications (ICISA), 2011 International Conference on 2011.

[16] A. Javed and A. Khokhar, "Frequent Pattern Mining on Message Passing Multiprocessor Systems," In Distributed and Parallel Databases, vol. 16, no. 3, pp. 321-334, 2004.

[17] S. Kotsiantis and D. Kanellopoulos, "Association Rules Mining: A Recent Overview," GETS International Transactions on Computer Science and Engineering, Vol.32(1), 2006, pp.71-82.

[18] H. Marghny and H. Refaat, "Hori-Vertical Distributed Frequent Itemsets Mining Algorithm on Heterogeneous Distributed Shared Memory System," IJCSNS International Journal of Computer Science and Network Security,VOL.10 No.11 , pp. 56–62 , November 2010.

[19] O. Zaïane, M. El-Hajj, and P. Lu, "Fast Parallel Association Rule Mining without Candidacy Generation," In Proceedings of the IEEE International Conference on Data Mining, 2001, pp. 665-668.

[20] M. Zaki, S. Parthasarath, and L. Wei, "A localized algorithm for parallel association mining," In Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, 1997, pp. 321–330.

[21] Q. Zhao and S. Bhowmick, "Association Rule Mining: A Survey," Technical Report, CAIS, Nanyang Technological University, Singapore, No. 2003116 , 2003.

[22] E. Zitzler, K. Deb, and L. Thiel, "An evolutionary Algorithm for Multi-objective Optimization," The Strength Pareto Approach, 1998.

 ISBN: 978-1-61208-247-9