PARMA: A Parallel Randomized Algorithm for Approximate Association Rules Mining in MapReduce

Matteo Riondato, Justin A. DeBrabant, Rodrigo Fonseca, and Eli Upfal
Dept. of Computer Science, Brown University
Providence, RI, USA
{matteo, debrabant, rfonseca, eli}@cs.brown.edu

ABSTRACT

Frequent Itemsets and Association Rules Mining (FIM) is a key task in knowledge discovery from data. As the dataset grows, the cost of solving this task is dominated by the component that depends on the number of transactions in the dataset. We address this issue by proposing PARMA, a parallel algorithm for the MapReduce framework, which scales well with the size of the dataset (as number of transactions) while minimizing data replication and communication cost. PARMA cuts down the dataset-size-dependent part of the cost by using a random sampling approach to FIM. Each machine mines a small random sample of the dataset, of size independent from the dataset size. The results from each machine are then filtered and aggregated to produce a single output collection. The output will be a very close approximation of the collection of Frequent Itemsets (FI's) or Association Rules (AR's) with their frequencies and confidence levels. The quality of the output is probabilistically guaranteed by our analysis to be within the user-specified accuracy and error probability parameters. The sizes of the random samples are independent from the size of the dataset, as is the number of samples. They depend on the user-chosen accuracy and error probability parameters and on the parallel computational model. We implemented PARMA in Hadoop MapReduce and show experimentally that it runs faster than previously introduced FIM algorithms for the same platform, while 1) scaling almost linearly, and 2) offering even higher accuracy and confidence than what is guaranteed by the analysis.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*; H.3.4 [**Systems and Software**]: Distributed Systems

General Terms

Algorithms, Experimentation, Performance, Theory

Keywords

MapReduce, Frequent Itemsets, Association Rules, Sampling

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.

1. INTRODUCTION

The discovery of (top-K) Frequent Itemsets and Association Rules (FIM) is a fundamental primitive in data mining and databases applications. The computational problem is defined in the general setting of a transactional dataset – a collection of transactions where each transaction is a set of items. With datasets increasing both in size and complexity, the computation for FIM faces scalability challenges in both space and time. Datasets have now reached the tens of terabytes scale and it is no longer reasonable to assume that such massive amounts of data can be easily processed by a single machine in a sequential fashion.

A typical exact algorithm scans the entire dataset, possibly multiple times, and stores intermediate counts of a large number of possible frequent itemsets candidates [2, 17]. The cost of these algorithms can be split in two independent components: the *scanning* cost and the *mining* cost. The scanning cost includes all operations that directly handle the transactions in the dataset, and scales with the *size* of the dataset, i.e., the number of such transactions. Examples include the scanning of the dataset to build the FP-Tree in FP-Growth [17] or to compute the actual frequencies of candidate frequent itemsets in APriori [1]. The mining cost refers to the operations in derived data structures and does not require access to the dataset. Examples include the operations performed on the FP-Tree once it has been generated, and the creation of candidate itemsets of length $i + 1$ at the end of phase i in APriori. This cost scales with the *complexity* of the dataset, i.e., the number of items, the number and distribution of frequent itemsets, and the underlying process that generated the transactions. It also depends on parameters given to the algorithm, such as the desired frequency threshold.

In this paper we are concerned with the scalability of FIM with respect to the size of the dataset, or the number of transactions. In many practical settings for FIM, the process generating the data changes very slowly or not at all, especially when compared to the data generation rate, therefore the number and frequency distribution of the frequent itemsets grows much slower than the size of the dataset. For example, the number of items available on the catalog of an e-commerce website grows much slower than the number of purchases by customers, each of which corresponds to a transaction. Therefore the scanning component grows faster than the mining one, and soon becomes dominant.

We introduce a randomized parallel algorithm for approximate frequent itemset mining, PARMA, that makes the scanning step of FIM embarassingly parallel, thus exhibiting near-linear speedup with the number of machines. PARMA combines random sampling and parallelization techniques in a novel fashion. It mines, in parallel, a set of small random samples and then filters and aggregates the collections of frequent itemsets or association rules obtained from each sample. Our work is orthogonal to other ap-

proaches, like PFP [18], which focuses on parallelizing the mining phase in order to decrease the corresponding component of the cost. Due to the use of random sampling, the output of PARMA is an approximation of the collection of FI's or AR's in the dataset, but leveraging on previous work [29], PARMA offers tight probabilistic guarantees on the quality of the approximated collections returned in output. In particular it guarantees that the output is an ε-approximation of the real collection with probability at least $1 - \delta$, where ε and δ are parameters specified by the user (see Section 3 for formal definitions). PARMA is designed on MapReduce [11], a novel parallel/distributed architecture that has raised significant interest in the research and industry communities. MapReduce is capable of handling very large datasets and efficiently executing parallel algorithms like PARMA.

To our knowledge PARMA is the first algorithm to exploit the combination of random sampling and parallelization for the task of Association Rules Mining. A number of previous works explored either parallel algorithms [4, 8, 12, 13, 22, 25, 30, 34] or random sampling [32, 35, 26, 28, 20, 29] for the FIM task, but the two approaches have been seen somewhat orthogonal until today. In PARMA, the disadvantages of either approach are evened out by the advantages of the other. In the spirit of *moving computation to the data* to minimize communication, we avoid data replication, and preserve the advantages of parallelization by using of multiple independent small random samples of the dataset which are mined in parallel and have only their results aggregated. Similarly, we are not subject to the inherent trade-off between the size of the random sample and the accuracy of the approximation that can be obtained from it, as PARMA would only have to mine more samples of the same size in parallel to get higher quality approximations.

Although PARMA is not the first algorithm to use MapReduce to solve the Association Rule Mining task, it differs from and enhances previous works [10, 14, 16, 18, 19, 33, 36] in two crucial aspects. First, it significantly reduces the data that is replicated and transmitted in the *shuffle* phase of MapReduce. Second, PARMA is not limited to the extraction of Frequent Itemsets but can also directly compute the collection of Association Rules in MapReduce. In previous works, association rules had to be created sequentially after the Frequent Itemsets had been computed in MapReduce.

We conducted an extensive experimental evaluation to test the relative performance, scalability and accuracy of PARMA across a wide range of parameters and datasets. Our results suggest that PARMA can significantly outperform exact mining solutions, has near-linear speedup, and, as data and nodes are scaled together, is able to achieve near constant runtimes. Also, our accuracy evaluation shows that PARMA consistently computes approximated collections of higher quality than what can be analytically guaranteed.

In this paper:

1. We present PARMA, the first randomized MapReduce algorithm for discovering approximate collections of frequent itemsets or association rules with near-linear speedup.
2. We provide analytical guarantees for the quality of the approximate results generated by the algorithm.
3. We demonstrate the effectiveness of PARMA on many datasets and compare the performance of our implementation to that of several exact FIM algorithms on MapReduce.

2. PREVIOUS WORK

The task of mining frequent itemsets is a fundamental primitive in computer science, with applications ranging from market basket analysis to network monitoring. The two most well known algorithms for extracting frequent itemsets from a dataset are APriori [2] and FP-growth [17].

The use of parallel and/or distributed algorithms for Association Rules Mining comes from the impossibility to handle very large datasets on a single machine. Early contributions in this area are presented in a survey by Zaki [34]. In recent years, the focus shifted to exploit architecture advantages as much as possible, such as shared memory [30], cluster architecture [4] or the massive parallelism of GPUs [13]. The main goal is to avoid communication between nodes as much as possible and minimize the amount of data that are moved across the network [8, 12, 22, 25].

The use of sampling to mine an approximation of the frequent itemsets and of association rules is orthogonal to the efforts for parallelizing frequent itemsets mining algorithm, but is driven by the same goal of making the mining of massive datasets possible. It was suggested in [23] almost as soon as the first efficient algorithms for Association Rules Mining appeared. Toivonen [32] presented the first algorithm to extract an approximation of the Frequent Itemsets using sampling. Many other works used different tools from probability theory to improve the quality of the approximations and/or reduce the size of sample. We refer the interested reader to [29] for a review of many contributions in this area.

The MapReduce [11] paradigm enjoys widespread success across both industry and academia. Research communities in many different fields uses this novel approach to distributed/parallel computation to develop algorithms to solve important problems in computer science [5, 6, 15, 21, 27]. Not only MapReduce can easily perform computation on very large datasets, but it is also extremely suited in executing embarassingly parallel algorithms which make a very limited use of communication. PARMA fits in this description so MapReduce is an appropriate choice for it.

A number of previous works [10, 14, 16, 18, 19, 33, 36] looked at adapting APriori and FP-growth to the MapReduce setting. Somewhat naively, some authors [10, 19, 33] suggest a distributed/parallel counting approach, i.e. to compute the support of every itemset in the dataset in a single MapReduce round. This algorithm necessarily incurs in a huge data replication, given that an exponential number of messages are sent to the reducers, as each transaction contains a number of itemsets that is exponential in its length. A different adaptation of APriori to MapReduce is presented in [16, Chap.4]: similarly to the original formulation of APriori, at each round i, the support for itemsets of length i is computed, and those that are deemed frequent are then used to generate candidate frequent itemsets of length $i + 1$, although outside of the MapReduce environment. Apart from this, the major downsides of such approach are that some data replication still occurs, slowing down the shuffling phase, and that the algorithm does not complete until the longest frequent itemset is found. Given that length is not known in advance, the running time of the algorithm can not be computed in advance. Also the entire dataset needs to be scanned at each round, which can be very expensive, even if it is possible to keep additional data structures to speed up this phase.

An adaptation of FP-Growth to MapRreduce called PFP is presented in [18]. First, a parallel/distributed counting approach is used to compute the frequent items, which are then randomly partitioned into groups. Then, in a single MapReduce round the transactions in the dataset are used to generate group-dependent transactions. Each group is assigned to a reducer and the corresponding group-dependent transactions are sent to this reducer which then builds the local FP-tree and the conditional FP-trees recursively, computing the frequent patterns. The group-dependent transactions are such that the local FP-trees and the conditional FP-trees built by different reducers are independent. This algorithm suffers from a

data replication problem: the number of group-dependent transactions generated for each single transaction is potentially equal to the number of groups. This means that the dataset may be replicated up to a number of times equal to the number of groups, resulting in a huge amount of data to be sent to the reducers and therefore in a slower synchronization/communication (*shuffle*) phase, which is usually the most expensive in a MapReduce algorithm. Another practical downside of PFP is that the time needed to mine the dependent FP-tree is not uniform across the groups. An empirical solution to this load balancing problem is presented in [36], although with no guarantees and by computing the groups outside the MapReduce environment. An implementation of the PFP algorithm as presented in [18] is included in Apache Mahout [3].

The authors of [14] presents an high-level library to perform various machine learning and data mining tasks using MapReduce. They show how to implement the Frequent Itemset Mining task using their library. The approach is very similar to that in [18], and the same observations apply about the performances and downsides of this approach.

3. DEFINITIONS

A *dataset* $\mathcal{D}$ is a collection of *transactions*, where each transaction τ is a subset of a ground set (alphabet) $\mathcal{I}$. There can be multiple identical transactions in $\mathcal{D}$. Members of $\mathcal{I}$ are called *items* and members of $2^{\mathcal{I}}$ are called *itemsets*. Given an itemset $A \in 2^{\mathcal{I}}$, let $T_{\mathcal{D}}(A)$ denote the set of transactions in $\mathcal{D}$ that contain A. The *support* of A, $\sigma_{\mathcal{D}}(A) = |T_{\mathcal{D}}(A)|$, is the number of transaction in D that contains A, and the *frequency* of A, $f_{\mathcal{D}}(A) = \frac{|T_{\mathcal{D}}(A)|}{|\mathcal{D}|}$, is the fraction of transactions in $\mathcal{D}$ that contain A.

A task of major interest in this setting is finding the *Frequent Itemsets with respect to a minimum frequency threshold*.

Definition 1. Given a *minimum frequency threshold* θ, $0 < \theta \leq 1$, the *Frequent Itemsets mining task with respect to* θ is finding all itemsets with frequency $\geq \theta$, i.e., the set

$$\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta) = \{(A, f_{\mathcal{D}}(A)) \ : \ A \in 2^{\mathcal{I}} \text{ and } f_{\mathcal{D}}(A) \geq \theta\}.$$

For the top-K Frequent Itemsets definition we assume a fixed *canonical ordering* of the itemsets in $2^{\mathcal{I}}$ by decreasing frequency in $\mathcal{D}$, with ties broken arbitrarily. We label the itemsets $A_1, A_2, \ldots, A_m$ according to this ordering and denote with $f_{\mathcal{D}}^{(K)}$ the frequency $f_{\mathcal{D}}(A_K)$ of the K-th most frequent itemset A_K. For a given K, with $1 \leq K \leq m$, the set of top-K Frequent Itemsets (with their respective frequencies) is defined as

$$\mathsf{TOPK}(\mathcal{D}, \mathcal{I}, K) = \mathsf{FI}(\mathcal{D}, \mathcal{I}, f_{\mathcal{D}}^{(K)}). \tag{1}$$

One of the main uses of frequent itemsets is in the discovery of *association rules*.

Definition 2. An *association rule* W is an expression "$A \Rightarrow B$" where A and B are itemsets such that $A \cap B = \emptyset$. The *support* $\sigma_{\mathcal{D}}(W)$ (resp. frequency $f_{\mathcal{D}}(W)$) of the association rule W is the support (resp. frequency) of the itemset $A \cup B$. The *confidence* $c_{\mathcal{D}}(W)$ of W is the ratio $\frac{f_{\mathcal{D}}(A \cup B)}{f_{\mathcal{D}}(A)}$ of the frequency of $A \cup B$ to the frequency of A.

Given a minimum frequency threshold θ and a minimum confidence level γ, we define the set $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta, \gamma)$ of triplets

$$(W, f_{\mathcal{D}}(W), c_{\mathcal{D}}(W))$$

s. t. W is an association rule with $f_{\mathcal{D}}(W) \geq \theta$ and $c_{\mathcal{D}}(W) \geq \gamma$.

In this work we are interested in computing well defined approximations of the above sets.

Definition 3. Given two parameters $\varepsilon_1, \varepsilon_2 \in (0, 1)$, an $(\varepsilon_1, \varepsilon_2)$-approximation of $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$ is a set $\mathcal{C} = \{(A, f_A, \mathcal{K}_A) \ : \ A \in 2^{\mathcal{I}}, f_A \in \mathcal{K}_A \subseteq [0, 1]\}$ of triplets $(A, f_A, \mathcal{K}_A)$ where f_A approximates $f_{\mathcal{D}}(A)$ and $\mathcal{K}_A$ is an interval containing f_A and $f_{\mathcal{D}}(A)$. $\mathcal{C}$ is such that:

1. $\mathcal{C}$ contains all itemsets appearing in $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$;
2. $\mathcal{C}$ contains no itemset A with frequency $f_{\mathcal{D}}(A) < \theta - \varepsilon_1$;
3. For every triplet $(A, f_A, \mathcal{K}_A) \in \mathcal{C}$, it holds
   (a) $|f_{\mathcal{D}}(A) - f_A| \leq \varepsilon_2$.
   (b) f_A and $f_{\mathcal{D}}(A)$ belong to $\mathcal{K}_A$.
   (c) $|\mathcal{K}_A| \leq 2\varepsilon_2$.

If $\varepsilon_1 = \varepsilon_2 = \varepsilon$ we refer to $\mathcal{C}$ as a ε-approximation of $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$.

This definition extends easily to the case of top-K frequent itemsets mining using the equivalence from (1). An $(\varepsilon_1, \varepsilon_2)$-approximation to $\mathsf{FI}\left(\mathcal{D}, \mathcal{I}, f_{\mathcal{D}}^{(K)}\right)$ is an $(\varepsilon_1, \varepsilon_2)$-approximation to $\mathsf{TOPK}(\mathcal{D}, \mathcal{I}, K)$.

For association rules, we have the following definition.

Definition 4. Given two parameters $\varepsilon_1, \varepsilon_2 \in (0, 1)$ an $(\varepsilon_1, \varepsilon_2)$-approximation of $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta, \gamma)$ is a set

$$\mathcal{C} = \{(W, f_W, \mathcal{K}_W, c_W, \mathcal{J}_W) \mid \text{AR } W, f_W \in \mathcal{K}_W, c_W \in \mathcal{J}_W\}$$

of tuples $(W, f_W, \mathcal{K}_W, c_W, \mathcal{J}_W)$ where f_W and c_W approximate $f_{\mathcal{D}}(W)$ and $c_{\mathcal{D}}(W)$ respectively and belong to $\mathcal{K}_W \subseteq [0, 1]$ and $\mathcal{J}_W \subseteq [0, 1]$ respectively. $\mathcal{C}$ is such that:

1. $\mathcal{C}$ contains all association rules appearing in $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta, \gamma)$;
2. $\mathcal{C}$ contains no association rule W with frequency $f_{\mathcal{D}}(W) < \theta - \varepsilon_1$;
3. For every tuple $(W, f_W, \mathcal{K}_W, c_W, \mathcal{J}_W) \in \mathcal{C}$, it holds $|f_{\mathcal{D}}(W) - f_W| \leq \varepsilon_2$ and $|\mathcal{K}_W| \leq 2\varepsilon_2$.
4. $\mathcal{C}$ contains no association rule W with confidence $c_{\mathcal{D}}(W) < \gamma - \varepsilon_1$;
5. For every tuple $(W, f_W, \mathcal{K}_W, c_W, \mathcal{J}_W) \in \mathcal{C}$, it holds $|c_{\mathcal{D}}(W) - c_W| \leq \varepsilon_2$ and $|\mathcal{J}_W| \leq 2\varepsilon_2$.

If $\varepsilon_1 = \varepsilon_2 = \varepsilon$ we refer to $\mathcal{C}$ as an ε-approximation of $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta, \gamma)$.

The following result from [29] is at the core of our algorithm for computing an ε-approximation to $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$. A similar result also holds for $\mathsf{TOPK}(\mathcal{D}, \mathcal{I}, K)$ [29, Lemma 3].

LEMMA 1. *[29, Lemma 1] Let $\mathcal{D}$ be a dataset with transactions built on an alphabet $\mathcal{I}$, and let d be the maximum integer such that $\mathcal{D}$ contains at least d transactions of size at least d. Let $0 < \varepsilon, \delta, \theta < 1$. Let $\mathcal{S}$ be a random sample of $\mathcal{D}$ containing $|\mathcal{S}| = \frac{2}{\varepsilon^2}\left(d + \log \frac{1}{\delta}\right)$ transactions drawn uniformly and independently at random with replacement from those in $\mathcal{D}$, then with probability at least $1 - \delta$, the set $\mathsf{FI}(\mathcal{S}, \mathcal{I}, \theta - \frac{\varepsilon}{2})$ is a $(\varepsilon, \varepsilon/2)$-approximation of $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$.*

For computing a ε-approximation to $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta)$, we make use of the following Lemma.

LEMMA 2. *[29, Lemma 6] Let $\mathcal{D}$ be a dataset with transactions built on an alphabet $\mathcal{I}$, and let d be the maximum integer such that $\mathcal{D}$ contains at least d transactions of size at least d. Let $0 < \varepsilon, \delta, \theta, \gamma < 1$ and let $\varepsilon_{\text{rel}} = \frac{\varepsilon}{\max\{\theta, \gamma\}}$. Fix $c > 4 - 2\varepsilon_{\text{rel}}$, $\eta = \frac{\varepsilon_{\text{rel}}}{c}$, and $p = \frac{1-\eta}{1+\eta}\theta$. Let $\mathcal{S}$ be a random sample of $\mathcal{D}$ containing $\frac{1}{\eta^2 p}(d \log \frac{1}{p} + \log \frac{1}{\delta})$ transactions from $\mathcal{D}$ sampled independently and uniformly at random. Then $\mathsf{AR}(\mathcal{S}, \mathcal{I}, (1 - \eta)\theta, \frac{1-\eta}{1+\eta}\gamma)$ is an $(\varepsilon, \varepsilon/2)$ approximation to $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta, \gamma)$.*

3.1 MapReduce

MapReduce is a programming paradigm and an associated parallel and distributed implementation for developing and executing parallel algorithms to process massive datasets on clusters of commodity machines [11]. Algorithms are specified in MapReduce using two functions, **map** and **reduce**. The input is seen as a sequence of ordered key-value pairs (k, v). The **map** function takes as input one such $(key, value)$ pair at a time, and can produce a finite multiset of pairs $\{(k_1, v_1), (k_2, v_2), \cdots\}$. Let $\mathcal{U}$ be the multiset union of all the multisets produced by the **map** function when applied to all input pairs. We can partition $\mathcal{U}$ into sets $\mathcal{U}_{\bar{k}}$ indexed by a particular key $\bar{k}$. $\mathcal{U}_{\bar{k}}$ contains all and only the values v for which there are pairs $(\bar{k}, v)$ with key $\bar{k}$ produced by the function **map** ($\mathcal{U}_{\bar{k}}$ is a multiset, so a particular value v can appear multiple times in $\mathcal{U}_{\bar{k}}$). The **reduce** function takes as input a key $\bar{k}$ and the multiset $\mathcal{U}_{\bar{k}}$ and produce another set $\{(k_1, v_1), (k_2, v_2), \cdots\}$. The output of **reduce** can be used as input for another (different) **map** function to develop MapReduce algorithms that complete in multiple *rounds*. By definition, the **map** function can be executed in parallel for each input pair. In the same way, the computation of the output of **reduce** for a specific key k^* is independent from the computation for any other key $k' \neq k^*$, so multiple copies of the **reduce** function can be executed in parallel, one for each key k. We denote the machines executing the **map** function as *mappers* and those executing the **reduce** function as *reducers*. The latter will be indexed by the key k assigned to them, i.e., reducer r processes the multiset $\mathcal{U}_r$. The data produced by the mappers are split by key and sent to the reducers in the so-called *shuffle* step. Some implementations, including Hadoop and the one described by Google [11], use sorting in the shuffle step to perform the grouping of map outputs by key. The shuffle is transparent to the algorithm designer but, since it involves the transmission of (possibly very large amount of) data across the network, can be very expensive.

4. ALGORITHM

In this section we describe and analyze PARMA, our algorithm for extracting ε-approximations of $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$, $\mathsf{TOPK}(\mathcal{D}, \mathcal{I}, \theta)$, and $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta, \gamma)$ from samples of a dataset $\mathcal{D}$ with probability at least $1 - \delta$. In this section we present the variant for $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$. The variants for the cases of $\mathsf{TOPK}(\mathcal{D}, \mathcal{I}, \theta)$ and $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta, \gamma)$ can be easily derived from the one we present here. We outline them in Section 4.4. Detailed presentations for them will appear in the full version of the paper.

4.1 Design

We now present the algorithmic design framework on which we developed PARMA and some design decisions we made for speeding up the computation.

Model. When developing solutions for any computational problem, the algorithm designer must always be aware of the trade-off between the available computational resources and the performance (broadly defined) of the algorithm. In the parallel computation setting, the resources are usually modeled through the parameters p and m, representing respectively the number of available processors that can run in parallel and the amount of local memory available to a single processor. In our case we will express m in terms of the number of transactions that can be stored in the main memory of a single machine. When dealing with algorithms that use random samples of the input, the performances of the algorithm are usually measured through the parameters ε and δ. The former represents the desired accuracy of the results, i.e., the maximum tolerable error (defined according to some distance measure) in the solution computed by the algorithm using the random sample when compared to an exact solution of the computational problem. The parameter δ represents the maximum acceptable probability that the previously defined error in the solution computed by the algorithm is greater than ε. The measure we will use to evaluate the performances of PARMA in our analysis is based on the concept of ε-approximation introduced in Definition 3.

Trade-offs. We are presented with a trade-off between the parameters ε, δ, p, and m. To obtain a ε-approximation with probability at least $1 - \delta$, one must have a certain amount of computational resources, expressed by p and m. On the other hand, given p and m, it is possible to obtain ε-approximations with probability at least $1-\delta$ only for values of ε and δ larger than some limits. By fixing any three of the parameters, it is possible to find the best value for the fourth by solving an optimization problem. From Lemma 1 we know that there is a trade-off between ε, δ, and the size w of a random sample from which it is possible to extract a $(\varepsilon, \varepsilon/2)$-approximation to $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$ with probability at least $1 - \delta$. If $w \leq m$, then we can store the sample in a single machine and compute the ε-approximation there using Lemma 1. For some combinations of values for ε and δ, though, we may have that $w > m$, i.e. the sample would be too large to fit into the main memory of a single processor, defeating one of the goals of using random sampling, that is to store the set of transactions to be mined in main memory in order to avoid expensive disk accesses. To address the issue of a single sample not fitting in memory, PARMA works on multiple samples, say N with $N \leq p$, each of size $w \leq m$ so that **1)** each sample fits in the main memory of a single processor and **2)** for each sample, it is possible to extract an $(\varepsilon, \varepsilon/2)$-approximation of $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$ from it with probability at least $1 - \phi$, for some $\phi > \delta$. In the first stage, the samples are created and mined in parallel and the so-obtained collections of Frequent Itemset are then aggregated in a second stage to compute the final output. This approach is a perfect match for the MapReduce framework, given the limited number of synchronization and communication steps that are needed. Each stage is performed in a single MapReduce round. The computational and data workflow of PARMA is presented in Figure 1, which we describe in detail in the following paragraphs.

Computing N and w. From the above discussion it should be clear that, once p, m, ε and δ have been fixed, there is a trade-off between w and N. In the MapReduce setting, often the most expensive operation is the movement of data between the mappers and the reducers in the shuffle phase. In PARMA, the amount of data to be shuffled corresponds to the sum of the sizes of the samples, i.e., Nw, and to the amount of communication needed in the aggregation stage. This second quantity is dependent on the number of frequent itemsets in the dataset, and therefore PARMA has no control over it. PARMA tries to minimize the first quantity when computing N and w in order to achieve the maximum speed. It is still possible to minimize for other quantities (e.g. ε or δ if they have not been fixed), but we believe the most effective and natural in the MapReduce setting is the minimization of the communication. This intuition was verified in our experimental evaluation, where communication proved to be the dominant cost. We can formulate the problem of minimizing Nw as the following Mixed Integer Non Linear Programming (MINLP) problem:

- **Variables:** non-negative integer N, real $\phi \in (0, 1)$,
- **Objective:** minimize $2N/\varepsilon^2(d + \log(1/\phi))$.

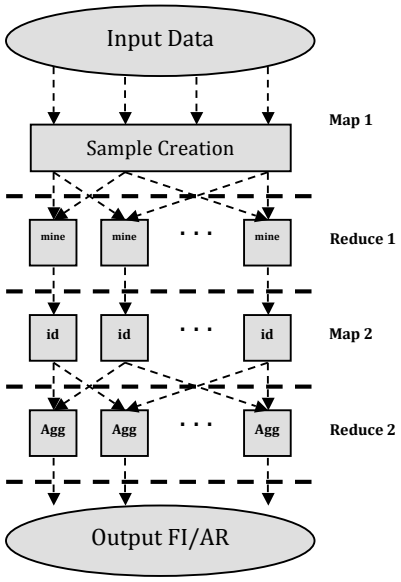

Figure 1: A system overview of PARMA. Ellipses represent data, squares represent computations on that data and arrows show the movement of data through the system.

- **Constraints:**

$$N \leq p \tag{2}$$

$$\phi \geq e^{-m\varepsilon^2/2+d} \tag{3}$$

$$N(1-\phi) - \sqrt{N(1-\phi)2\log(1/\delta)} \geq N/2 + 1 \tag{4}$$

Note that, because of our requirement **2)** on w, the sample size w is directly determined by ϕ through Lemma 1, so the trade-off is really between N and ϕ, while w does not appear in the above problem. Since ϕ is a probability we restrict its domain to the interval $(0, 1)$, but it must also be such that the single sample size w is at most m, as required by **1)** and expressed by Constraint (3). The limit to the number of samples N is expressed by Constraint (2). The last constraint (4) is a bit more technical and the need for it will be evident in the analysis of the algorithm. Intuitively, it expresses the fact that an itemset must appear in a sufficiently high fraction (at least 1/2, possibly more) of the collections obtained from the samples in the first stage in order to be included in the output collection. Due to the integrality constraint on N, this optimization problem is not convex, although when the constraint it is dropped the feasibility region is convex, and the objective function is convex. It is then relatively easy and fast to find an integer optimal solution to the problem using a global MINLP solver like BARON [31].

4.2 Description

In the following paragraphs we give a detailed description of PARMA. The reader is also referred to Figure 1 for a schematic representation of PARMA's data/computational workflow.

Stage 1: Sampling and Local Mining. Once ϕ, w and N have been computed, PARMA enters the first MapReduce round to create the N samples (phase Map 1 in Figure 1) and mine them (Reduce 1). We see the input of the algorithm as a sequence

$$(1, \tau_1), (2, \tau_2), \cdots, (|\mathcal{D}|, \tau_{|\mathcal{D}|}),$$

where the τ_i are transactions in $\mathcal{D}$. In the Map phase, the input of the **map** function is a pair (tid, τ), where tid is a natural from 1 to $|\mathcal{D}|$ and τ is a transaction in $\mathcal{D}$. The map function produces in output a pair $(i, (\ell_\tau^{(i)}, \tau))$ for each sample $\mathcal{S}_i$ containing τ. The value $\ell_\tau^{(i)}$ denotes the number of times τ appears in $\mathcal{S}_i$, $1 \leq i \leq N$. We use random sampling with replacement and ensure that all samples have size w, i.e., $\sum_{\tau \in \mathcal{D}} \ell_\tau^{(i)} = w, \forall i$. This is done by computing (serially) how many transactions each mapper must send to each sample. In the Reduce phase, there are N reducers, with associated key i, $1 \leq i \leq N$. The input to reducer i is $(i, \mathcal{S}_i)$, $1 \leq i \leq N$. Reducer i mines the set $\mathcal{S}_i$ of transactions it receives using an exact sequential mining algorithm like Apriori or FP-Growth and a lowered minimum frequency threshold $\theta' = \theta - \varepsilon/2$ to obtain $\mathcal{C}_i = \mathsf{FI}(\mathcal{S}_i, \mathcal{I}, \theta')$. For each itemset $A \in \mathcal{C}_i$ the Reduce function outputs a pair $(A, (f_{\mathcal{S}_i}(A), [f_{\mathcal{S}_i}(A) - \varepsilon/2, f_{\mathcal{S}_i}(A) + \varepsilon/2]))$.

Stage 2: Aggregation. In the second round of MapReduce, PARMA aggregates the result from the first stage to obtain a ε-approximation to $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$ with probability at least $1 - \delta$. The Map phase (Map 2 in Figure 1) is just the identity function, so for each pair

$$(A, (f_{\mathcal{S}_i}(A), [f_{\mathcal{S}_i}(A) - \varepsilon/2, f_{\mathcal{S}_i}(A) + \varepsilon/2])$$

in the input the same pair is produced in the output. In the Reduce phase (Reduce 2) there is a reducer for each itemset A that appears in at least one of the collections $\mathcal{C}_j$ (i.e., $\forall A$ such that there is a $\mathcal{C}_j$ containing a pair related to A). The reducer receives as input the itemset A and the set $\mathcal{F}_A$ of pairs

$$(f_{\mathcal{S}_i}(A), [f_{\mathcal{S}_i}(A) - \varepsilon/2, f_{\mathcal{S}_i}(A) + \varepsilon/2])$$

for the samples $\mathcal{S}_i$ such that $A \in \mathcal{C}_i$. Now let

$$R = N(1-\phi) - \sqrt{N(1-\phi)2\log(1/\delta)}. \tag{5}$$

The itemset A is declared *globally frequent* and will be present in the output if and only if $|\mathcal{F}_A| \geq R$. If this is the case, PARMA computes, during the Reduce phase of the second MapReduce round, the estimation $\tilde{f}(A)$ for the frequency $f_{\mathcal{D}}(A)$ of the itemset A in $\mathcal{D}$ and the confidence interval $\mathcal{K}_A$. The computation for $\tilde{f}(A)$ proceeds as follows. Let $[a_A, b_A]$ be the *shortest* interval such that there are at least $N - R + 1$ elements from $\mathcal{F}_A$ that belong to this interval. The estimation $\tilde{f}(A)$ for the frequency $f_{\mathcal{D}}(A)$ of the itemset A is the central point of this interval:

$$\tilde{f}(A) = a_A + \frac{b_A - a_A}{2}$$

The confidence interval $\mathcal{K}_A$ is defined as

$$\mathcal{K}_A = \left[a_A - \frac{\varepsilon}{2}, b_A + \frac{\varepsilon}{2}\right].$$

The output of the reducer assigned to the itemset A is

$$(A, (\tilde{f}(A), \mathcal{K}_A)).$$

The output of PARMA is the union of the outputs from all reducers.

4.3 Analysis

We have the following result:

LEMMA 3. *The output of the PARMA is an ε-approximation of $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$ with probability at least $1 - \delta$.*

PROOF. For each sample $\mathcal{S}_i$, $1 \leq i \leq N$ we define a random variable X_i that takes the value 1 if $\mathcal{C}_i = \mathsf{FI}(\mathcal{S}_i, \mathcal{I}, \theta')$ is a $(\varepsilon, \varepsilon/2)$-approximation of $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$, $X_i = 0$ otherwise. Given

our choices of w and θ', we can apply Lemma 1 and have that $\Pr(X_i = 1) \geq 1 - \phi$. Let $Y = \sum_{r=1}^{N} X_r$ and let Z be a random variable with binomial distribution with parameters N and $1 - \phi$. For any constant $Q < N(1 - \phi)$ we have

$$\Pr(Y \leq Q) \leq \Pr(Z \leq Q) \leq e^{-N(1-\phi)(1-\frac{Q}{N(1-\phi)})^2/2},$$

where the last inequality follows from an application of the Chernoff bound [24, Chap. 4]. We then have, for our choice of ϕ and N and for $Q = R$ (defined in Eq. (5)), that with probability at least $1 - \delta$, at least R of the collections $\mathcal{C}_i$ are $(\varepsilon, \varepsilon/2)$-approximations of $\mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$. Denote this event as $\mathcal{G}$. For the rest of the proof we will assume that $\mathcal{G}$ indeed occurs.

Then $\forall A \in \mathsf{FI}(\mathcal{D}, \mathcal{I}, \theta)$, A belongs to at least R of the collections $\mathcal{C}_i$, therefore a triplet $(A, \tilde{f}(A), \mathcal{K}_A)$ will be in the output of the algorithm. This means that Property 1 from Def. 3 holds.

Consider now any itemset B such that $f_{\mathcal{D}}(B) < \theta - \varepsilon$. By definition of $(\varepsilon, \varepsilon/2)$-approximation we have that B can only appear in the collections $\mathcal{C}_i$ that are not $(\varepsilon, \varepsilon/2)$-approximations. Given that $\mathcal{G}$ occurs, then there are at most $N - R$ such collections. But from Constraint (4) and the definition of R in (5), we have that $N - R < R$, and therefore B will not be present in the output of PARMA, i.e. Property 2 from Def. 3 holds.

Let now C be any itemset in the output, and consider the interval $S_C = [a_C, b_C]$ as computed by PARMA. S_C contains at least $N - R + 1$ of the $f_{\mathcal{S}_i}(C)$, otherwise C would not be in the output. By our assumption on the event $\mathcal{G}$, we have that at least R of the $f_{\mathcal{S}_i}(C)$'s are such that $|f_{\mathcal{S}_i}(C) - f_{\mathcal{D}}(C)| \leq \varepsilon/2$. Then there is an index j such that $|f_{\mathcal{S}_j}(C) - f_{\mathcal{D}}(C)| \leq \varepsilon/2$ and such that $f_{\mathcal{S}_j}(C) \in S_C$. Given also that $f_{\mathcal{S}_j}(C) \geq a_C$, then $f_{\mathcal{D}}(C) \geq a_C - \varepsilon/2$, and analogously, given that $f_{\mathcal{S}_j}(C) \leq b_C$, then $f_{\mathcal{D}}(C) \leq b_C + \varepsilon/2$. This means that

$$f_{\mathcal{D}}(C) \in \left[a_C - \frac{\varepsilon}{2}, b_C + \frac{\varepsilon}{2}\right] = \mathcal{K}_C, \tag{6}$$

which, together with the fact that $\tilde{f}_C \in \mathcal{K}_C$ by construction, proves Property 3.b from Def. 3. We now give a bound to $|S_C| = b_C - a_C$. From our assumption on the event $\mathcal{G}$, there are (at least) R values $f_{\mathcal{S}_i}(C)$ such that $|f_{\mathcal{S}_i}(C) - f_{\mathcal{D}}(C)| \leq \varepsilon/2$, then the interval $[f_{\mathcal{D}}(C) - \varepsilon/2, f_{\mathcal{D}}(C) + \varepsilon/2]$ contains (at least) R values $f_{\mathcal{S}_i}(C)$. Its length ε is an upper bound to $|S_C|$. Then the length of the interval $\mathcal{K}_C = [a_C - \varepsilon/2, b_C + \varepsilon/2]$ is at most 2ε, as requested by Property 3.c from Def. 3. From this, from (6), and from the fact that $\tilde{f}(C)$ is the center of this interval we have $|\tilde{f}(C) - f_{\mathcal{D}}(C)| \leq \varepsilon$, i.e., Property 3.a from Def. 3 holds. □

4.4 Top-K Frequent Itemsets And Association Rules

The above algorithm can be easily adapted to computing, with probability at least $1 - \delta$, ε-approximations to $\mathsf{TOPK}(\mathcal{D}, \mathcal{I}, K)$ and to $\mathsf{AR}(\mathcal{D}, \mathcal{I}, \theta, \gamma)$. The main difference is in the formula to compute the sample size w (Lemma 1), and in the process to extract the local collections from the samples. The case of top-K is presented in [29] and is a minor modification of Lemma 1, while for the association rule case we can use Lemma 2. These are minimal changes to the version of PARMA presented here, and the modified algorithms guarantee the same levels of accuracy and confidence.

5. IMPLEMENTATION

The entire PARMA algorithm has been written as a Java library for Hadoop, the popular open source implementation of MapReduce. Because all experiments were done using Amazon Web Service (AWS) Elastic MapReduce, the version of Hadoop used was 0.20.205, the highest supported by AWS. The use of Java makes possible future integration with the Apache Mahout parallel machine learning library [3]. Mahout also includes an implementation of PFP [18] that we used for our evaluation of PARMA.

In PARMA, during the mining phase (i.e. during the reducer of stage 1), any frequent itemset or association rule mining algorithm can be used. We wanted to compare the performances of PARMA against PFP which only produces frequent itemsets, therefore we chose to use a frequent itemset mining algorithm instead of an association rule mining algorithm. Again, this choice was merely for ease of comparison with existing parallel frequent itemset mining algorithms as no such algorithms for association rule mining exist. While there are many frequent itemset mining algorithms available, we chose the FP-growth implementation provided by [7]. We chose FP-growth due to its relative performance superiority to other Frequent Itemsets mining algorithms. Additionally, since FP-growth is the algorithm that PFP has parallelized and uses internally, the choice of FP-growth for the mining phase in PARMA is appropriate for a more natural comparison.

We also compare PARMA against the naive distributed counting algorithm (DistCount) for computing frequent itemsets. In this approach, there is only a single MapReduce iteration. The map breaks a transaction τ into its powerset $\mathcal{P}(\tau)$ and emits key/value pairs in the form $(A, 1)$ where A is an itemset in $\mathcal{P}(t)$. The reducers simply count how many pairs they receive for each itemset A and output the itemsets with frequency above the minimum frequency threshold. This is similar to the canonical wordcount example for MapReduce. However, because the size of the powerset is exponential in the size of the original transaction (specifically $2^{|\tau|}$, where $|\tau|$ denotes the number of items in a given transaction), this algorithm incurs massive network costs, even when combiners are used. This is very similar to the algorithms presented in [10, 19, 33]. We have built our own implementation of DistCount in Java using the Hadoop API.

6. EXPERIMENTAL EVALUATION

We evaluate the performance of PARMA using Amazon's Elastic MapReduce platform. We used instances of type *m1.xlarge*, which contain roughly 17GB of memory and 6.5 EC2 compute units. For data, we created artificial dataset using the synthetic data generator from [9]. This implementation is based on the generator described in [2], which can be parameterized to generate a wide range of data. We used two distinct sets of parameters to generate the datasets: the first set, shown in Table 1, for the experiments comparing PARMA and the distributed counting algorithm (DistCount), and the second set, shown in Table 2, for the experiments comparing PARMA and PFP. The parameters were chosen to mimic real-world datasets on which PARMA would be run. For a full description of the relevant parameters, we refer the reader to [2]. The reason we needed two distinct datasets is that DistCount did not scale to the larger dataset sizes, as the amount of data it generates in the map phase grows exponentially with the length of the individual transactions in the dataset. We found that DistCount would run out of memory using datasets with longer transactions, and we had to generate datasets with both shorter and less transactions for its comparisons.

Because PARMA is an approximate algorithm, the choice of accuracy parameters ε and δ are important, as is θ, the minimum frequency at which itemsets were mined. In all of our experiments, $\varepsilon = 0.05$ and $\delta = 0.01$. This means that the collection of itemsets mined by PARMA will be a 0.05-approximation with probability 0.99. In practice, we show later that the results are much more accurate than what this. For all experiments other than the minimum

number of items	1000
average transaction length	5
average size of maximal potentially large itemsets	5
number of maximal potentially large itemsets	5
correlation among maximal potentially large itemsets	0.1
corruption of maximal potentially large itemsets	0.1

Table 1: Parameters used to generate the datasets for the runtime comparison between DistCount and PARMA in Figure 2.

number of items	10000
average transaction length	10
average size of maximal potentially large itemsets	5
number of maximal potentially large itemsets	20
correlation among maximal potentially large itemsets	0.1
corruption of maximal potentially large itemsets	0.1

Table 2: Parameters used to generate the datasets for the runtime comparison between PFP and PARMA in Figure 2.

frequency performance comparison in Figure 4 and for the accuracy comparison in Figures 7 and 8, θ was kept constant at 0.1.

Due to space limitations, we do not report the results of the experiments to evaluate the performances of PARMA as the parameters of the optimization problem change.

6.1 Performance Analysis

For the performance analysis of PARMA, we analyze the relative performance against two exact FIM algorithms on MapReduce, *DistCount* and *PFP*, on a cluster of 8 nodes. We also provide a breakdown of the costs associated with each stage of PARMA.

Figure 2 (top) shows the comparison between PARMA and DistCount. For DistCount, longer itemsets affect runtime the most, as the number of key/value pairs generated from each transaction is exponential in the size of the transaction. This is not to say that more transactions does not affect runtime, just that the length of those transactions also has a significant impact. Because of this, it is possible to have datasets with fewer transactions but with more "long" transactions that take longer to mine. This effect is seen in the first three datasets (1-3 million). Of course, since these datasets were generated independently and with the same parameters, this was purely by chance. However, as the number of transactions continues to increase, the exponential growth in the number of intermediate key/value pairs is seen by the sharp increase in runtime. While we tried to test with a dataset with 6 million transaction, DistCount ran out of memory. The lack of ability to handle either long individual transactions or a large number of transactions in a dataset limits DistCount's real-world applicability.

For the performance comparison with PFP, 10 datasets were generated using parameter values from Table 2 and ranging in size from 10 to 50 million transactions. The results are shown in Figure 2 (bottom). For every dataset tested, PARMA was able to mine the dataset roughly 30-55% faster than PFP. The reason for the relative performance advantage of PARMA is twofold. The first (and primary) reason is that for larger datasets the size of the dataset that PARMA has sampled (and mined) is staying the same, whereas PFP is mining more and more transactions as the dataset grows. The second reason is that as the dataset grows, PFP is potentially duplicating more and more transactions as it assigns transactions to groups. A transaction that belongs to multiple groups is sent to multiple reducers, resulting in higher network costs.

The most important aspect of the comparison of PFP to PARMA

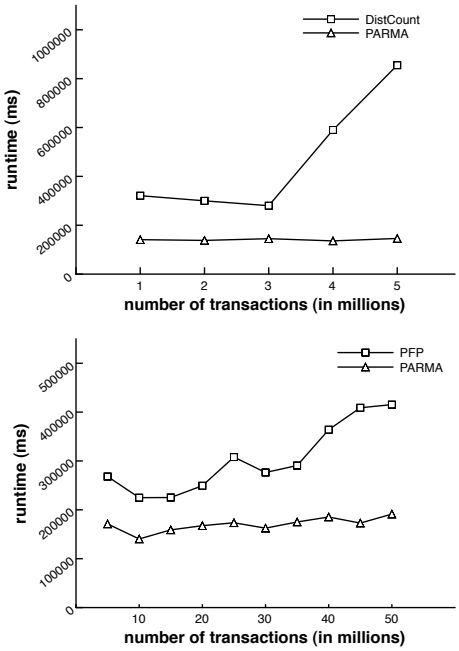

Figure 2: A runtime comparison of PARMA with DistCount (top) and PFP (bottom).

is that the runtimes as data grows are clearly diverging due to the reasons discussed above. While 50 million transactions is very sizable, it is not hard to imagine real-world datasets with transactions on the order of billions. Indeed, many point-of-sale datasets would easily break this boundary. In these scenarios a randomized algorithm such as PARMA would show increasing performance advantages over exact algorithms such as any of the standard non-parallel algorithms or PFP, which must mine the entire dataset. At that scale, even transmitting that data over the network (several times in the case of PFP) would become prohibitive.

To understand the performance of PARMA it is important to analyze the runtimes at each of the various stages in the algorithm. To do this, we have implemented runtime timers at very fine granularities throughout our algorithm. The timers' values are written to Hadoop job logs for analysis. This breakdown allows us to not only analyze the overall runtime, but also the sections of the algorithm whose runtimes are affected by an increase in data size. In Figure 3, a breakdown of PARMA runtimes is shown for each of the six segments of the algorithm, which include a map, shuffle and reduce phase for each of the two stages. Due to space limitations, we only show the breakdown for a subset of the datasets we tested. We observed the same patterns for all datasets. This breakdown demonstrates several interesting aspects of PARMA. First, the cost of the mining local frequent itemsets (stage 1, reduce) is relatively constant. For many frequent itemset mining implementations, this cost will grow with the size of the input. This is not the case in PARMA, because local frequent itemset mining is being done on constant-sized sample of the input. Indeed another interesting observation, as expected, is that the only cost that increases as sample size increases is the cost of sampling (stage 1, map). This is because in order to be sampled the input data must be read, so larger input data means larger read times. In practice, this cost is minimal and grows linearly with the input, hence it will never be prohibitive,

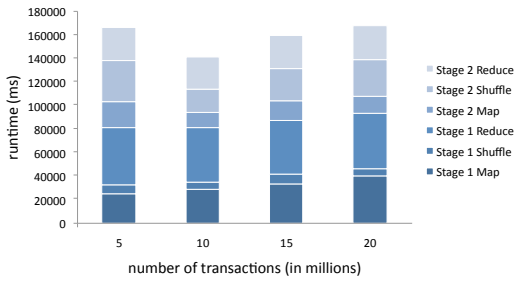

Figure 3: A comparison of runtimes of the map/reduce/shuffle phases of PARMA, as a function of number of transactions. Run on an 8 node Elastic MapReduce cluster.

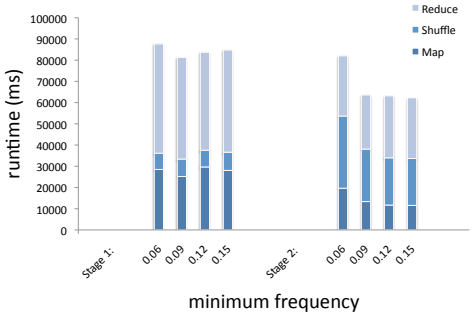

Figure 4: A comparison of runtimes of the map/reduce/shuffle phases of PARMA, as a function of minimum frequency. Clustered by stage. Run on an 8 node Elastic MapReduce cluster.

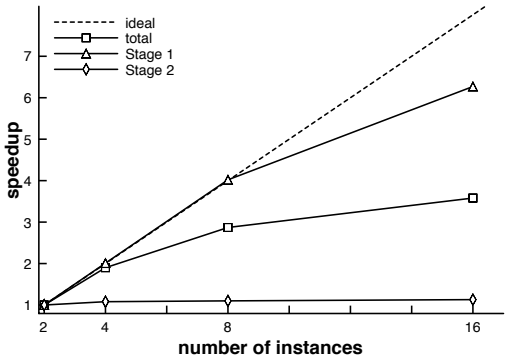

Figure 5: The speedup analysis of PARMA, broken down by stages.

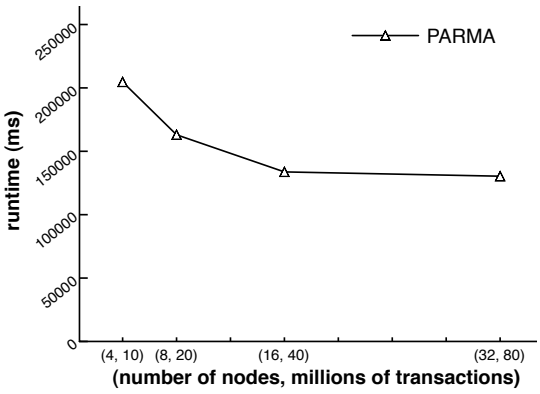

Figure 6: The scalability of PARMA as both data and cluster size are increased.

especially considering all other current algorithms must read the entire input data at least once, and in many cases multiple times.

There is one outlier in the graph, which is the dataset with 5 million transactions. Because each dataset was independently generated, it is possible for a dataset to have a larger number of frequent itemsets than other datasets, even if it has less transactions. This is the case with the 5 million transaction dataset, which takes longer to run mine for both PARMA and PFP due to the relatively greater number of frequent itemsets.

Figure 4 shows the breakdown of PARMA runtimes as the minimum frequency at which the data is mined at is changed. Data size was kept constant at 10 million transactions. Minimum frequency is used by the local frequent itemset mining algorithm to prune itemsets; itemsets below the minimum frequency are not considered frequent, nor is any superset since a superset must, by definition, contain the not frequent set and therefore cannot be frequent itself. Intuitively, a lower minimum frequency will mean more frequent itemsets are produced. Other than a runtime increase in the local frequent itemset mining phase (stage 1, reduce), the effects of this can be seen in the stage 2 shuffle phase as well, as there is more data to move across the network. Still, the added costs of mining with lower frequencies are relatively small.

6.2 Speedup and Scalability

To show the speedup of PARMA, we used a two-nodes cluster as the baseline. Because PARMA is intended to be a parallel algorithm, the choice of a two-nodes cluster was more appropriate than the standard single node baseline. For the dataset, we used a 10 million transaction database generated using the parameters in Table 2. The results are shown in Figure 5. The three lines on this graph represent the relative speedup of both stage 1 and stage 2 as well as the overall PARMA algorithm. The graph indicates that stage 1 is highly parallelizable and follows a near-ideal speedup for up to 8 nodes, after which a slight degradation of speedup occurs. There are two reasons for this slight degradation. In the map phase of stage 1, due to an implementation decision in Hadoop, the smallest unit of data that can be split is one HDFS block. As we continue to add more nodes to the cluster, we may have more available map slots than HDFS data blocks, resulting in some slots being unused. Theoretically, this could be fixed by allowing smaller granularity splitting in Hadoop. Another cause of the slightly sub-ideal speedup in stage 1 is from the reducer. Because the data in this experiment was held constant, the slight degradation in speedup as more than 8 nodes were added was a result of an inefficient over-splitting of transaction data. If each reducer in stage 1 is mining a very small subset of the transactions, the overhead of building the FP-tree begins to dominate the cost of mining the FP-tree. This is because the cost of mining the FP-tree is relatively fixed. Thus, we can "over-split" the data by forcing the reducer to build a large FP-tree only to mine a small set of transactions. For larger samples, the size of the cluster where speedup degradation begins to occur would also increase, meaning PARMA would continue to scale.

Also, as is clearly visible in the graph, the sub-ideal overall speedup is due largely to the poor speedup of stage 2. Stage 2 is bound almost entirely by the communication costs of transmitting the local frequent itemsets from stage 1 to the reducers that will

do the aggregation. Because the amount of local frequent itemsets does not change as more nodes are added, the communication for this stage does not change. What does change is the number of itemsets each node must aggregate. During the reduce phase, each node is assigned a set of keys. All key/value pairs emitted from the map phase are sent to the reducer assigned their respective key. The reducer is in charge of aggregating the values and emitting one aggregate value per key assigned to it. As more reducers are added to the cluster, each reducer will have fewer keys assigned to it, and therefore must aggregate across fewer values, resulting in faster aggregation. The small but existent positive change in the line for stage 2 is a result of this slight speedup of the reduce phase.

Figure 6 depicts the scalability of PARMA as both the size of the dataset (i.e. number of transactions) and the number of nodes in the cluster are increased. The data and nodes are scaled proportionally so that the ratio of data to nodes remains constant across all experiments. This result shows that as nodes and data are increased proportionally, the total runtime actually begins to decrease for larger datasets. This is because as nodes are added to the cluster, the runtime of the Stage 1 reducer (FIM) is decreased while the relative costs of the Stage 1 mapper and Stage 2 remain the same. There is a leveling off of the runtime between the 40M and 80M datasets, which can be explained using Amdhal's law; because only portions of the algorithm are parallelizable, there is a theoretical maximum speedup that is possible. Still, the constant runtime as data is increased demonstrates PARMA's potential scalability to real-world cluster and dataset sizes.

6.3 Accuracy

The output of PARMA is a collection of frequent itemsets which approximates the collection one can obtain by mining the entire dataset. Although our analysis shows that PARMA offers solid guarantees in terms of accuracy of the output, we conducted an extensive evaluation to assess the actual performances of PARMA in practice, especially in relation to what can be analytically proved.

We compared the results obtained by PARMA with the exact collection of itemsets from the entire dataset, for different values of the parameters ε, δ, and θ, and for different datasets. A first important result is that in all the runs, the collection computed by PARMA was indeed a ε-approximation to the real one, i.e., all the properties from Definition 3 were satisfied. This fact suggests that the confidence in the result obtained by PARMA is actually greater than the level $1-\delta$ suggested by the analysis. This can be explained by considering that we had to use potentially loose theoretical bounds in the analysis to make it tractable.

Given that all real frequent itemsets were included in the output, we then focused on how many itemsets with real frequency in the interval $[\theta-\varepsilon, \theta)$ were included in the output. It is important to notice that these itemsets would be *acceptable* false positives, as Definition 3 does not forbid them to be present in the output. We stress again that the output of PARMA never contained *non-acceptable* false positives, i.e. itemsets with real frequency less than the minimum frequency threshold θ. The number of acceptable false positives included in the output of PARMA depends on the distribution of the real frequencies in the interval $[\theta-\varepsilon, \theta)$, so it should not be judged in absolute terms. In Table 3 we report, for various values of θ, the number of real frequent itemsets (i.e., with real frequency at least θ, the number of acceptable false positives (AFP) contained in the output of PARMA, and the number of itemsets with real frequency in the interval $[\theta-\varepsilon, \theta)$, i.e., the maximum number of acceptable false positives that may be contained in the output of PARMA (Max AFP). These numbers refers to a run of PARMA on (samples of) the 10M dataset, with $\varepsilon = 0.05$ and $\delta = 0.01$. It is evident that PARMA does a very good job in filtering out even acceptable false positives, especially at lower frequencies, when their number increases. This is thanks to the fact that an itemset is included in the output of PARMA if and only if it appears in the majority of the collections obtained in the first stage. Itemsets with real frequencies in $[\theta-\varepsilon, \theta)$ are not very likely to be contained in many of these collections.

θ	Real FI's	Output AFP's	Max AFP's
0.06	11016	11797	201636
0.09	2116	4216	10723
0.12	1367	335	1452
0.15	1053	299	415

Table 3: Acceptable False Positives in the output of PARMA

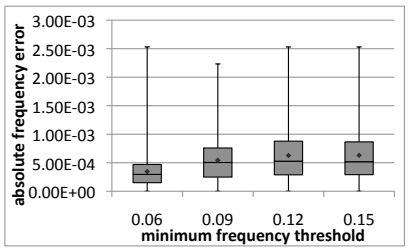

Figure 7: Error in frequency estimations as frequency varies.

We conducted an evaluation of the accuracy of two other components of the output of PARMA, namely the estimated frequencies for the itemsets in the output and the width of the confidence bounds for these estimations. In Figure 7 we show the distribution of the absolute error in the estimation, i.e. $|\tilde{f}(X) - f_{\mathcal{D}}(X)|$ for all itemsets X in the output of PARMA, as θ varies. The lower end of the "whisker" indicates the minimum error, the lower bound of the box corresponds to the first quartile, the segment across the box to the median, and the upper bound of the box to the third quartile. The top end of the whisker indicates the maximum error, and the central diamond shows the mean. This figure (and also Figure 8) shows the values for a run of PARMA on samples of the 10M dataset, with $\varepsilon = 0.05$ and $\delta = 0.01$. We can see that even the maximum values are one order of magnitude smaller than the threshold of 0.05 guaranteed by the analysis, and many of the errors are two or more orders of magnitude smaller. It is also possible to appreciate that the distribution of the error would be heavily concentrated in a small interval if the maximum error were not so high, effectively an outlier. The fact that the average and the median of the error, together with the entire "box" move down as the minimum frequency threshold decrease can be explained by the fact that at lower frequencies more itemsets are considered, and this makes the distribution less susceptible to outliers. Not only this is a sign of the high level of accuracy achieved by PARMA, but also of its being consistently accurate on a very large portion of the output.

Finally, in Figure 8 we show the distribution of the widths of the confidence intervals $\mathcal{K}(A)$ for the frequency estimations $\tilde{f}(A)$ of the itemsets A in the output of PARMA. Given that $\varepsilon = 0.05$, the maximum allowed width was $2\varepsilon = 0.1$. It is evident from the figures that PARMA returns much narrower intervals, of size almost ε. Moreover, the distribution of the width is very concentrated, as shown by the small height of the boxes, suggesting that PARMA is extremely consistent in giving very high quality confidence intervals for the estimations. We state again that in all runs of PARMA in our tests, all the confidence intervals contained the estimation

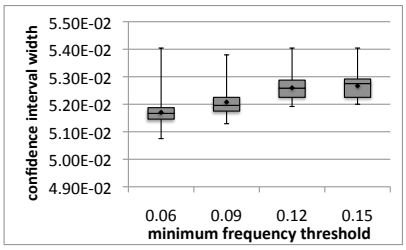

Figure 8: Width of the confidence intervals as frequency varies.

and the real frequency, as requested by Definition 3. As seen in the case of the estimation error, the distribution of the widths shifts down at lower thresholds θ. This is motivated by the higher number of itemsets in the output of PARMA at those frequencies. Their presence makes the distribution more robust to outliers. We can conclude that PARMA gives very narrow but extremely accurate confidence intervals across the entirety of its output.

This analysis of the accuracy of various aspects of PARMA's output shows that PARMA can be very useful in practice, and the confidence of the end user in the collections of itemsets and estimations given in its output can be even higher than what is guaranteed by the analysis.

7. CONCLUSIONS

In this paper, we have described PARMA, a parallel algorithm for mining quasi-optimal collections of frequent itemsets and association rules in MapReduce. We showed through theoretical analysis that PARMA offers provable guarantees on the quality of the output collections. Through experimentation on a wide range of datasets ranging in size from 5 million to 50 million transactions, we have demonstrated a 30-55% runtime improvement over PFP, the current state-of-the-art in exact parallel mining algorithms on MapReduce. Empirically we were able to verify the accuracy of the theoretical bounds, as well as show that in practice our results are orders of magnitude more accurate than is analytically guaranteed. Thus PARMA is an algorithm that can scale to arbitrary data sizes while simultaneously providing nearly perfect results.

8. ACKNOWLEDGMENTS

The work of Riondato, DeBrabant, and Upfal was supported in part by NSF award IIS-0905553.

9. REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. SIGMOD '93.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. VLDB '94.

[3] Apache Mahout. `http://mahout.apache.org/`.

[4] G. Buehrer, S. Parthasarathy, S. Tatikonda, T. Kurc, and J. Saltz. Toward terabyte pattern mining: an architecture-conscious solution. PPoPP '07.

[5] F. Chierichetti, R. Kumar, and A. Tomkins. Max-cover in Map-Reduce. WWW '10.

[6] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. R. Bradski, A. Y. Ng, and K. Olukotun. Map-Reduce for machine learning on multicore. NIPS '06.

[7] F. Coenen. `http://www.cxc.liv.ac.uk/~frans/KDD/Software/FPgrowth/fpGrowth.html`

[8] S. Cong, J. Han, J. Hoeflinger, D. Padua. A sampling-based framework for parallel data mining. PPoPP '05.

[9] L. Cristofor. ARTool. `http://www.cs.umb.edu/~laur/ARtool/`, 2006.

[10] J.-D. Cryans, S. Ratté, and R. Champagne. Adaptation of APriori to MapReduce to build a warehouse of relations between named entities across the web. DBKDA '10.

[11] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *CACM*, 51(1):107–113, 2008.

[12] M. El-Hajj and O. Zaiane. Parallel leap: large-scale maximal pattern mining in a distributed environment. ICPADS '06.

[13] W. Fang, K. K. Lau, M. Lu, X. Xiao, C. K. Lam, Y. Yang, B. He, Q. Luo, P. V. Sander, and K. Yang. Parallel data mining on graphics processors. Technical Report 07, The Hong Kong University of Science & Technology, 2008.

[14] A. Ghoting, P. Kambadur, E. Pednault, and R. Kannan. NIMBLE: a toolkit for the implementation of parallel data mining and machine learning algorithms on MapReduce. KDD '11.

[15] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the MapReduce framework. *CoRR*, abs/1101.1902, 2011.

[16] S. Hammoud. *MapReduce Network Enabled Algorithms for Classification Based on Association Rules*. PhD thesis, Brunel University, 2011.

[17] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29:1–12, May 2000.

[18] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. PFP: Parallel FP-Growth for query recommendation. RecSys '08.

[19] L. Li and M. Zhang. The strategy of mining association rule based on cloud computing. BCGIN '11.

[20] Y. Li and R. Gopalan. Effective sampling for mining association rules. AI '04.

[21] J. Lin and M. Schatz. Design patterns for efficient graph algorithms in MapReduce. MLG '10.

[22] L. Liu, E. Li, Y. Zhang, Z. Tang. Optimization of frequent itemset mining on multiple-core processor. VLDB '07.

[23] H. Mannila, H. Toivonen, and I. Verkamo. Efficient algorithms for discovering association rules. KDD '94.

[24] M. Mitzenmacher and E. Upfal. *Probability and computing*. Cambridge University Press, 2005.

[25] E. Ozkural, B. Ucar, and C. Aykanat. Parallel frequent item set mining with selective item replication. *IEEE Trans. on Paral. and Distrib. Sys.*, 22(10):1632–1640, 2011.

[26] S. Parthasarathy. Efficient progressive sampling for association rules. ICDM '02.

[27] A. Pietracaprina, G. Pucci, M. Riondato, F. Silvestri, and E. Upfal. Space-round tradeoffs for MapReduce computations. ICS' 12.

[28] A. Pietracaprina, M. Riondato, E. Upfal, and F. Vandin. Mining top-K frequent itemsets through progressive sampling. *Data Min. and Knowl. Disc*, 21:310–326, 2010.

[29] M. Riondato and E. Upfal. Efficient discovery of association rules and frequent itemsets through sampling with tight performance guarantees. *CoRR*, abs/1111.6937v3, 2012.

[30] J. Ruoming, Y. Ge, and G. Agrawal. Shared memory parallelization of data mining algorithms: techniques, programming interface, and performance. *IEEE Trans. on Knowl. and Data Engin.*, 17(1):71–89, 2005.

[31] N. V. Sahinidis and M. Tawarmalani. *BARON 9.0.4: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2010.

[32] H. Toivonen. Sampling large databases for association rules. VLDB '96.

[33] X. Y. Yang, Z. Liu, and Y. Fu. MapReduce as a programming model for association rules algorithm on Hadoop. ICIS '10.

[34] M. Zaki. Parallel and distributed association mining: a survey. *IEEE Concurrency*, 7(4):14 –25, 1999.

[35] M. Zaki, S. Parthasarathy, W. Li, and M. Ogihara. Evaluation of sampling for data mining of association rules. RIDE '97.

[36] L. Zhou, Z. Zhong, J. Chang, J. Li, J. Huang, and S. Feng. Balanced parallel FP-Growth with MapReduce. YC-ICT '10.