# The State of Distributed Data Mining
## [ECS265 Project Report]

Michael Byrd Conny Franke

UC Davis, Davis CA 95616, USA
{byrd,franke}@cs.ucdavis.edu

**Abstract.** Data mining is known to be among the hottest topics in E-science. New methods for mining vast amounts of heterogeneous data from several data sources are emerging all the time. In this paper, we explore two of the most important data mining tasks in a distributed environment. We showcase some of the most important properties and algorithms of distributed frequent itemset mining and distributed clustering. Essentially two surveys for the price of one, here you will learn about the state of the art in distributed clustering and frequent itemset mining. Several of the most popular algorithms are explained and broken down to key points.

## 1 Introduction

Data Mining is now, as much as ever, a necessity in today's e-science environment. The problem comes quickly when we realize that our sources of data are growing quickly at a linear rate while the majority of our data mining algorithms take at least an $n^2$ time frame. At the rate our data is growing (millions to billions of records with hundreds of fields) even a linear time frame is growing quickly impossible. Databases have been distributed for quite some time in order to help divide the work or add redundancy. Data mining and, in particular, clustering have traditionally been done outside of a database. The current implementation of most data mining suites (WEKA, ADaM) operate directly on a file structure which is outside of the database. This involves shipping the data (possibly from multiple sources) to a single destination where all the processing takes place in a single processing computer. This mode of operation will no longer suffice as the rate of data growth is faster than the growth of network throughput. It becomes necessary to explore the mining of whole datasets efficiently in a distributed setting.

Distributed data mining requires an architecture which is completely different from the one used in centralized approaches. In a distributed environment, the architecture must facilitate to pay careful attention to distributed resources of data, computing, and communication [20]. The different architectures for centralized and distributed data mining are shown in figure 1, which is taken from [20]. The figure clearly shows that in a distributed setting, processing should be done

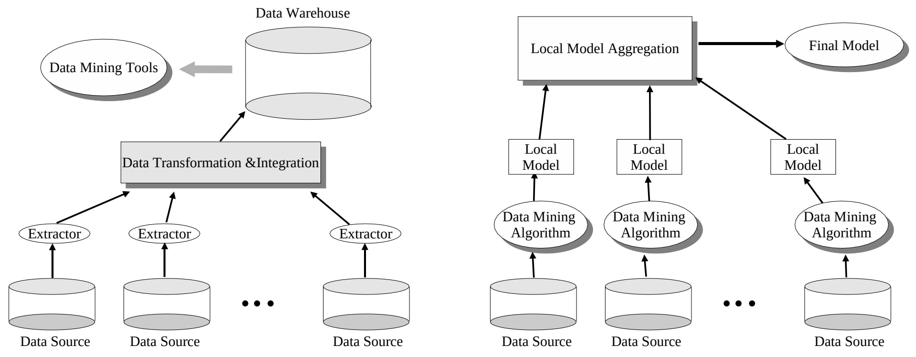

**Fig. 1.** Left: traditional data-warehouse based architecture for data mining. Right: architecture for distributed data mining. Source: [20]

on different nodes generating several local models. These models are then aggregated to form a global model representing the mining result of the entire dataset.

There are three major categories that data mining algorithms fall under, these are classification, frequent itemset mining, and clustering. Each is useful in its own right. In this paper, we investigate two of these data mining techniques in detail, namely clustering and frequent itemset mining.

First we explore the concepts related to the clustering of distributed data and summarize a few of the broader algorithms. We first attempt to define a set of rules that can measure the quality of an algorithm in a distributed setting. Following this we summarize some of the more encapsulating distributed clustering algorithms such that we get something of a family classification. Most other algorithms seem to fit into one of these families. Following our clustering chapter, we begin to explore frequent itemset mining. we give an overview of the general goal of the technique and the issues arising in the distributed case. A summary of a thorough paper survey will be given as well. We also identify open problems in both research areas and outline possible solutions or next steps.

The rest of this paper is organized as follows: Section 2 gives an introduction to distributed clustering and section 3 lists types of clustering algorithms and presents an example algorithm for each type. In section 4 we give an introduction to distributed frequent itemset mining. Representative algorithms for this data mining task are presented in section 5. In section 6 we describe open problems and possible future work in both research areas. Section 7 concludes the paper by summarizing our findings and future goals.

## 2 Introduction to Distributed Clustering

The idea of data mining is inherently to look at all the data and make statements about broad trends within it. Having distributed data is often a fact of life and must be dealt with and as such we must be able to analyze effectively. Unless stated otherwise, in this paper I assume that the distribution has already been decided on and that the query is written completely independent it. I think this is a fair assumption because we cannot assume we are given any natural advantage when asked to cluster distributed data. I think this is also valid because if you already have your data at a single source, there are several single pass algorithms that can give a much better approximation in much less time than would be required to ship the data. This is true when talking about large datasets anyway. I do present a couple of top down type ideas, but they will be explicitly defined as such.

There are a couple of crucial ideas that must be taken into account. The first of which is the idea of minimizing data transmission. If our methodology requires too much interaction throughout the clustering process then the distributed process will have much more ground to make up against just shipping all the data to a single source. A second point to consider is data privacy. It is entirely possible that we might have a situation (example forth coming) such that we want to get a clustering from a much larger set of data than we have from several sources. This is certainly a noble goal, but can conflict if the data is at all sensitive. Data privacy is an important attribute to keep in mind. We may choose an algorithm that can be processed locally at the source, then only passes back necessary statistics to another local source such that the local source can recreate the cluster pattern. This can be done using only the abstraction of the cluster, not necessarily exposing any single tuple directly through communication. An important consideration as well is how the abstraction of the cluster is handled. At some point every algorithm must transmit data for it to be accumulated. The question becomes how well they communicate these abstract data types, this is generally directly related to how they are formed. Local completeness is also an attribute we must keep in mind. If we are willing to use a distributed setting for our clustering we must know in advance if it will give us the same quality of clusters as another algorithm would if run on a single system. These are currently the best accepted measures for the quality of a distributed clustering methodology. In the next section we will go into each in more detail.

## 2.1 Data Privacy

Consider that you might be working in a hospital, doing research on many of the patients there. You are running a clustering algorithm on the hospitals patients which is returning adequate results, but the number of patients might not be enough to establish a real trend. You might write to another hospital and medical school asking to cluster their records, searching for a deeper broader package. This request would almost surely be denied due to the need to protect patient privacy. A hospital, financial institute, or secret government agency does not care to share any of its private data but might be willing to run an algorithm on its own data, such that only an abstraction of the cluster is returned.

This requirement can negate the need for algorithm efficiency. If a single source is unwilling to share any of its actual tuples, then we are forced to eliminate the naive plan of shipping all tuples. We must find a way of communicating how a local source contributes to the global cluster scheme without revealing any data from the local source. I'll give examples of several algorithms that present novel means of insuring data privacy. This is primarily a direct trait of how they abstract the cluster and how well the clusters are summarized.

## 2.2 Cluster Abstraction

An idea that helps is thinking of the clustering algorithm as a mapping. We give it a set of points as input and it then returns a set of clusters as output. We cannot call it a function as a single set of inputs can return various clusters depending on some random factors (as with K-means). The point of clustering is such that we can represent a large number of points into a single cluster that suits them well. We want a cluster to summarize the points within it as a grouping. There are a number of ways to deal with this process, as can be seen just in standard clustering. Results for a cluster can vary erratically as we will see ahead, but some common plans include: list of members, representative point, statistics and a radius, outer hull, and a functional representation. Those are a handful of the representations you will see in centralized clustering algorithms. Even intuitively some of these methodologies are a horrible idea for a distributed setting. Later examples of specific algorithms will give some illumination to the choices of abstract representation.

Some algorithms[12] utilize member representatives. By choosing a representative you lose a lot of information. Your hope is that in a manner similar to K-means you will be able to aggregate or summarize several points into a single point that some how captures all the information. The adaptive method of K-means does exactly the same thing as the centralized K-means does, however this limits the clusters to being circular in shape. By choosing a representative that is in the center and within a given radius of the rest of the points in the cluster we are able to consolidate all the information into that single point. This is fine for methods where the cluster is created by a single point, but certainly has weaknesses. We should also consider what happens when we use a density or grid based approach.

## 2.3 Minimize Data Transmission

The naive idea here for most all data mining suites is such that we just transmit all the data to the same site then run a standard single clustering technique there. This is often not feasible at all, as the dataset are so large they can barely be recorded, much less transferred to a single site and recorded temporarily there. The idea then must be to work on several local components and have them merge in some way or another. This directly implies that we can not send each point from each partition with membership, so we must abstract the cluster. There seem to be several classes of cluster abstraction for several different types of planning.

### 2.4 Local Completeness

An important consideration to keep in mind is that we cannot always control the distribution of the data. Keeping that in mind we need to either be forewarned or be sure that performing our clustering operation will provide for as accurate a clustering as transferring all the data to a central computer. This is to say, if we have a small cluster spread out to a single point on every distributed surface we should recognize it as a cluster in the global clustering.

## 3 General Listings of Types of Clustering Algorithms

Here we go over some of the more archetypal clustering algorithms that were included in the readings. I have tried to grab a few of the unique algorithms as most other algorithms are similar to one of these categories.

K-means is something of the most general methodology for clustering. It has traditionally been the most adaptable of the clustering algorithms as its concept of cluster abstraction is fairly standard. The (slightly more than naive) approach used by Kriegel et al. [15] and the more traditional distributed Kmeans will be presented here.

### 3.1 K-Means

To start out the Kmeans algorithm [7] only has to be moderately modified to work on a global level. We start out choosing our $K$ cluster centers randomly among all the random data points. The cluster centers are sent to every local representative and a local K means clustering takes place. Each local machine gathers it statistics about membership within its own clusters. The generation of these sufficient statistics are how even this naive algorithm fulfills a number of the rules above. The statistics are those sufficient to add up local pieces to recreate the whole. After this step each of the statistics are transmitted a central controller.

By representing each cluster as a set of gathered statistics we mostly satisfy the data privacy rule by not giving any single point of data away except under very specific circumstances.[⋆] By reducing any number of points into a single cluster we are reducing (although perhaps not minimizing) the data transferred. The downside is that we have to send this statistics over and over again until convergence is achieved. This is not assured to be a very quick process and depending on what transfer time is like it could end up being faster to just ship the data. Certainly, this methodology leaves a lot to be desired.

Datta et al.[6] use a similar methodology only remove the need for a central server. The by removing the need for a central point they allow for a far more

[⋆] It is entirely possible that you will have a K big enough such that some of the clusters could consist of a single point. In this case, the sufficient statistics are going to be a direct copy of the single tuple of data.

dynamic network. This algorithm is more tailored towards P2P and sensor networks. A non stable network such that any data that is available is considered the complete set. While this methodology is interesting, it is not really the object of this review.

### 3.2 Model Based

Kriegel et al. [15] develop this methodology a little bit more. On the local level we are based on EM (Expectation Maximization) clustering [17]. This is just like K-means except that we may use more functions to decide on the final clustering. So we start out with each local system processing only its own pieces. This methodology differs because we do not take to transferring partial results. Instead we take our local EM clustering and model each cluster as a sum of Gaussian functions. This is a handy methodology used again in other algorithms. There is also a measure of privacy that we employ. Similar to the previous algorithm, if there are too few actual data points represented by the local gaussian equation we may be able to decipher too much about the original data points. Therefore this algorithm gives a specific privacy algorithm to determine (and optionally change) whether the functions being used are giving too much information. These functions are then transferred to a global collection point when then combines the functions to give global information about the probability density of the global picture. This information is then sent to each local source who can then reevaluate this data in respect to this, or merely use the new information. The idea of using several probability functions to represent a cluster is also well-developed by Klusch et al.[14]

This methodology has problems that are not unique to distributed clustering. It has a standard problem that some call ‘strong connection’ such that two large clusters with a small densely connected component can end up being in the same cluster even though they should not be. There is also the weakness that the number of clusters is still a parameter. On the plus side, however, it no longer forces spherical type clusters which of great help to accuracy.

### 3.3 Density Grid

Density Based approaches[18] seemed to be among the most successful in a distributed clustering environment. This specific algorithm attempts to make the CLIQUE[1] algorithm more appropriate to a distributed setting. They improve a couple of problems that were in the original to make it better suited to follow the great rules of distributed clustering. They start out by scanning each of the attributes in the user specified query. Rather than setting some global setting for the size of each grid square they decide on it dynamically based on the statistics. This makes a lot of sense because depending on the distribution of the data bins need to be variable. This comes at a relatively small cost later when we must determine relative density of adjacent grid elements.

This is a very strong algorithm which yields strong clusters of even results. However, it is based on a top down methodology where it is assumed that we have the data centralized in a single repository then we distribute it (either to other processors or nodes). This algorithm is quite nice because it represents the clusters as a filling of a grid. This means, because of the dynamic gridding, that the cluster will be fine grained in heavy density areas and courser in areas of lower population.

### 3.4 Hierarchy

Another Archetypal distributed clustering methodology is presented by Johnson and Kargupta [13] in the form of a hierarchical algorithm. This uses a methodology similar to the previous in that it uses a density gridding type approach.

It uses distributed adaptation of single link seems to do a good job creating clusters through the use of dendograms and creating a tight bound on the minimum and maximum distance between the clusters. By tight bounding the distances at any local point, we can then transmit only the TID (any unique identifier for an object) along with it's necessary statistics for determining minimum distances for merging. These fairly minimal statistics can give us a lot of information for choosing which set of clusters to combine. The global model is kept relatively simple in combination by use of the *reducibility property*[13]. To summarize it simply, it says that two clusters combined into one cannot be any closer to a third cluster(assuming you are always taking the closest pair next) than the minimum of the two parts. This helps greatly when creating the global model.

### 3.5 Extend DBScan

Density based algorithms continue to be popular because of the additive property of densities. If we grid off the space of the clustering query we can add each local density to combine into a single estimate for the global subspace. At first Januzaj et al. [12] create the DBDC (Density Based Distributed Clustering) algorithm to make DB Scan a distributed friendly algorithm. Then Januzaj et al. [11] heroically proceed to extend their own DBDC algorithm to work better that same year. The algorithm starts out with each local branch using the DBscan algorithm to create a local clustering arrangement. The algorithm decides, perhaps in a counterintuitive move, to then use a member representative for each cluster. I found this choice odd as the beauty of DB Scan is it's ability to create clusters of irregular shape rather than just spherical ones. The member representatives are then shipped back to a single source with some statistics to a global repository (thereby violating the idea of data privacy) where a new scan is then conducted and the global information is dispatched.

### 3.6 Cluster Aggregation

When dealing with cluster aggregation[9]we are talking about taking several sets of potentially similar clusters and finding a clustering set up that is the least disagreeable to them all. This paper is quite clever and does not really need any single clustering methodology but rather can consolidate the results of several. This is generally useful for vertically split data where data privacy is demanded. This is because each set of data all refer to the same cluster so as long as there is a unique identifier for all tuples (which there must be for vertical partitioning). They each cluster their own aspects of the data and then can be queried regarding cluster agreement. The idea is that we take these various clustering schemes, a mapping for each item into a group, and attempt to merge them into a single plan. In this process we minimize the amount that all the clusters disagree with the final clustering. A clustering is said to agree with another for a single data item if they are sorted to the same grouping. While this algorithm is great for data privacy it cannot operate on horizontally fragmented data nor does it do very well with data transmission. There is a lot of data sent because we must either query for every single point or send a table regarding the unique identifier and cluster id of every data item.

## 4 Introduction to Distributed Frequent Itemset Mining

### 4.1 Overview and Objectives

In general, the objective of frequent itemset mining is to find all subsets of items which occur frequently together in a given set of transactions. Much research in the area of frequent itemset mining on a single machine has been done in the past. Starting with the Apriori algorithm proposed by Agrawal and Srikant [3], research has gone a long way in optimizing the performance of frequent itemset mining algorithms.

In the case of a finite number of transactions, the main performance criteria is the number of passes an algorithm has to perform over the entire dataset. Algorithms exists, e.g. the frequent pattern tree approach and the fp-growth method by Han et al. [10], that need only two passes over the data to determine the correct and complete set of frequent itemsets for a given frequency threshold $\sigma$.

When mining data streams, which opposed to a fixed set of transactions may contain an infinite number of transactions, the nature of the appropriate algorithms changes. Frequent itemset mining algorithms on data streams are inherently one-pass algorithms. Due to space and runtime constraints, those algorithms focus on determining an approximate result set rather than the correct and complete one. The first such algorithm, Lossy Counting, was published by Manku and Motwani [16]. Most other frequent itemset mining algorithms on data streams are either based on Lossy Counting, e.g. Chang and Lee's sliding window approach [4], or use the frequent pattern tree approach as Giannella et al. do in their work [8]. In the case of data streams, the main performance criteria for frequent itemset mining algorithms is the size of the maximal error $\epsilon$ in the approximate mining result. More precisely one could state this criteria as the ratio between the space requirements of the synopsis data structure and the magnitude of the guaranteed error threshold.

In a distributed setting the objectives of frequent itemset mining algorithms become more complex. The cost of communication between all sites participating in the distributed data mining is one of the main issues in this case. The communication overhead grows with the number of partitions. Schuster and Wolff state in [22] that some existing algorithms do not scale well with the number of partitions, thus motivating the need for a reduction of the overall communication between sites and a thorough communication cost analysis when designing such distributed algorithms.

Additional objectives in the distributed setting include data distribution techniques and load balancing. In a setting where the data is not distributed initially, an efficient strategy of how to distribute the data among the available sites is crucial for leveraging the advantages induced by the distributed environment. And since properties of the data may vary between different sites and thus resource requirements may differ from one site to another, load balancing during the processing of the data is an important consideration as well.

### 4.2 Data Distribution

There are two possible motivations for distributed processing of a data mining task. First, the data may be distributed initially, as in the domains of large companies, network traffic analysis and sensor networks. There, it may be beneficial to process the data on the local sites and only ship the data mining results, instead of shipping all data to one site and do the data mining on one node.

Second, if the dataset is tremendous, then the distributed computation of smaller portions of the data may yield a better performance than using a single centralized approach and mining the whole dataset at once. But as we mentioned in section 4.1 already, the necessary communication between the different computing sites has a huge impact on the improvements that can be gained in performance by distributing the frequent itemset mining task.

As in frequent itemset mining on a single node, the properies of the data partitions on each side significantly influence the run-time of the data mining algorithm (see e.g. [23, 28]). Some partitions may contain only very few frequent itemsets whereas others contain many frequent itemsets and thus take longer to process the data. So if the data distribution is highly skewed on different sites, the time required to get a mining result will vary. It is therefore an important step to balance the workload of every node in oder to achieve the best speed-up possible when using a distributed data mining approach.

In the first scenario mentioned the distribution of the data is already defined, since the individual datasets on each node represent a partitioning of the overall dataset being processed. Thus, not much can be done to fix initially skewed data distributions. However, dynamic load balancing mechanisms can be deployed while mining the data just like in the second scenario.

In the second scenario, where the data is initially on one site, the decision of how to partition the data is crucial for exploiting the benefits of distributed computing optimally. As mentioned above, dynamic load balancing, e.g. [25], during the processing of the data can be deployed to keep CPU and/or memory requirements nearly uniform over all nodes.

Data can be split horizontally and vertically, but according to Zaki's survey in 1999 [28] most papers assume a horizontal fragmentation, where each portion contains a subset of all transactions. Zaki's claim seems to be still true today. Only a few publication, like [21, 30], assume a vertical fragmentation, where each item is associated with a list of TIDs, which is a list of all transactions containing this item.

There are only very few publication that deal with data distribution as preprocessing step for frequent itemset mining specifically. One example for an algorithm that uses sophisticated partitioning strategies to achieve near optimal balancing between processors is the distributed version of the fp-growth algorithm [27].

However, most papers either assume distributed data initially, e.g. [26], or they distribute the given dataset arbitrarily among the nodes as in [23]. We are convinced that a sophisticated algorithm for partitioning a dataset in such a way that facilitates the distributed frequent itemset mining can help improve memory requirements and run-time benefits induced by the distributed processing even more.

## 4.3 Classification Criteria

Frequent itemset mining algorithms can be categorized according to different criteria. The most important ones are briefly described in the following paragraphs.

***Architecture*** The existing distributed frequent itemset mining algorithms assume different architectures. A cluster of shared-nothing machines is a simple and cost-efficient solution, since standard computers can be used and the system is therefore easily extendable. Most algorithms (a counterexample is DDM) for shared-nothing environments were shown to not scale well [24] and suffer from the fact that they have to scan the database as many times as the number of items in the largest frequent itemset.

Another common system is a shared-memory architecture, where each processor has equal access to the system's memory. Data mining algorithms assuming this architecture are easy to implement, but suffer from limited scalability due to the finite bandwidth of the shared memory. In addition, these systems are not as easily extendable as a shared-nothing architecture.

Some algorithms also use an agent-based approach, usually assuming a supervisor agent. This special agent controls the whole mining process by supervising all local agents.

***Bottom-up vs. Top-down Mining*** The traditional frequent itemset mining approach is level-based. In each round the global set of frequent $k$-itemsets and local sets of candidates for frequent $(k+1)$-itemsets are computed, where $k$ is starting from 1. A frequent $k$-itemset is an itemset with exactly $k$ items. Thus, round-based algorithms require as many iterations as the number of items in the largest frequent itemset. Since frequency counts of the local frequent $k$-itemsets have to be exchanged after each round, this method requires synchronization of all nodes and a relatively high number of messages. This method is a bottom-up approach since in each level larger frequent itemsets are computed based on the result of the previous round.

The size of the set of candidates generated at each site is the most important factor in the amount of necessary communication. One possibility to reduce the number of candidates is that each site first individually generates the local set of maximal frequent itemsets (frequent itemsets that are no subset of another frequent itemset). From this, the set of global maximal frequent itemsets (MFI) can be computed. Using this set, all frequent itemsets in the entire dataset can be enumerated without generating infrequent candidates of frequent itemsets. This is because of the *downward closure property*, which essentially says that each subset of a frequent itemset is again a frequent itemset. Thus, if the set of maximal frequent itemsets is known, then all frequent itemsets can be inferred from this. Some algorithms deploy this top-down approach of first generating the MFI set instead of a level-based approach. A top-down approach usually requires less communication since the local models are exchanged only after the local mining is finished, and not after every round.

***Mining Static vs. Dynamic Datasets*** In the first years of frequent itemset mining, one underlying assumption always was that the processed dataset is static. Thus, if changes in a database should be reflected in the data mining result, the frequent itemset mining was repeated for the whole database. With the emerging of highly dynamic datasets and data streams, the need for incremental algorithms grew. Incremental algorithms are capable of mining only the delta-portion of the dataset and incorporating these result in the overall mining result. Only a few incremental distributed frequent itemset mining algorithms exist, but their number will surely grow as will the need for them due to settings like sensor networks and network traffic monitors.

***Complete vs. Heuristic Mining*** Traditionally, frequent itemset mining algorithms aim at determining the complete set of frequent itemsets in the given database. This approach is used by the majority of distributed frequent itemset mining algorithms. However, some algorithms deploy sampling or other data reduction mechanisms to decrease the computational workload. Usually an error-bound is given to define the maximal possible error in the mining result.

Just like the approximate mining algorithms using the $\epsilon$-bound in the centralized case, heuristic algorithms in the distributed case are used to process rapidly changing dynamic datasets, where traditional approaches are too slow.

***Load Balancing*** Since distributed frequent itemset mining algorithms partition the data among the different nodes, they all inherently deploy a static initial load balancing. However, almost all algorithms fail to do dynamic load balancing during the data mining process. This may be due to the additional computation and communication cost imposed by the process of discovering a load imbalance and redistributing parts of the dataset. In general, we can assume that dynamic load balancing is only useful in shared-memory architectures, since otherwise the communication overhead induced by data movement exceeds the benefit of the load balancing.

## 5 Past Work in Distributed Frequent Itemset Mining

In this summary of past work on distributed frequent itemset mining we concentrate on only a few algorithms that are either fundamental to this research area or represent the current state of the art with respect to one or more of the criteria mentioned in section 4.3. A discussion of more algorithms can be found in the overview papers presented in section 5.1.

Frequent itemset mining is one of the two main steps necessary to discover association rules in a set of transactions or a data stream. Thus, in many cases the overall mining goal does not require the knowledge of the frequent itemsets but the association rules determined from them. Therefore the following section not only considers pure frequent itemset mining algorithm but also association rule mining algorithms that use the notion of frequent itemsets to compute a result.

### 5.1 Overview Papers

There exist several excellent survey and overview papers summarizing the work that has been done in the area of distributed frequent itemset and association rule mining. Since the research in this area is currently not very active, these papers still present the current state of the art in frequent itemset and association rule mining well.

A fairly old survey by Zaki [28] gives a good summary of existing association rule mining algorithms. Zaki categorizes the algorithms according to different criteria, gives examples of how they work and includes a big section about open problems. Many of these open problems remain until today. The paper also contains a good discussion of the different architectures used.

An overview paper is also written by Zaki [29]. In this paper, Zaki gives a very thorough categorization of distributed data mining algorithms and provides lots of examples of algorithms in each category. This paper provides a great overview of the field and its content is in big parts still fitting the current state of the art. Naturally, the content of this paper overlaps with [28], as many facts that apply to data mining algorithms in general are also true for association rule mining which is an important data mining task.

A more recent overview is given by Park et al. in 2002 [20]. Although they consider data mining in general, many aspects in the paper apply to frequent itemset mining. The paper details on data distribution and data pre-processing and provides a good overview of issues in data mining systems. Specifically, the impact of the underlying architecture on data mining is described thoroughly.

The section about distributed association rule mining gives a concise description of the basic concepts and techniques, but doesn't mention many existing algorithms in detail.

### 5.2 Distributed Apriori Algorithms

***Count Distribution*** Many distributed frequent itemset mining algorithms use an Apriori-based approach. The simplest one was published by Agrawal and Shafer in 1996 [2] where they proposed a distributed version of the Apriori algorithm. The algorithm is called *Count Distribution* and assumes a shared-nothing architecture.

In this round based algorithm, each node first computes the candidates for the frequent $k$-itemsets of its local data, starting with the frequent 1-itemsets, i.e. items. Then all nodes communicate their candidates together with the frequency values to all other nodes. From these information each node can now determine individually which itemsets are frequent in the overall data. The nodes then start the next round and compute all candidate $(k+1)$-itemsets based on the set of frequent $k$-itemsets they just obtained.

The communication cost of the Count Distribution algorithm depend on the length of the largest frequent itemset but are kept small by the fact that only candidates and their frequencies are sent.

***Data Distribution*** In the same paper as Count Distribution, Agrawal and Shafer also propose the *Data Distribution* algorithm [2] where all nodes compute disjoint sets of candidates. But because of a big communication overhead this algorithm performs much worse than Count Distribution.

***FDM*** Cheung et al. also proposed a version of distributed Apriori in the same year as Agrawal and Shafer did. In [5] they present the *FDM* algorithm (fast distributed mining of association rules), which differs from Count Distribution in the content of the messages sent between the nodes. This algorithm also assumes a shared-nothing architecture.

***DDM*** The *Distributed Decision Miner* (DDM) [22] was presented by Schuster and Wolff in 2001. This algorithm belongs to the group of Apriori-based algorithms assuming a shared-nothing architecture as well. Here, after local frequency counts are computed on each node, the nodes perform a distributed decision protocol in each round in order to determine the set of globally frequent itemsets. In this protocol, a shared public and a private hypothesis is maintained at each site. Nodes can choose to publish their local frequency counts of itemsets. According to the protocol, if no node decides to publish its frequency count of a certain itemset, then the global hypothesis regarding this itemset is correct, i.e., it is known to the public hypothesis if this itemset is globally frequent.

Because of this protocol, DDM is very communication-efficient. It is also more scalable and adapts better to data skewness than other Apriori-based algorithms.

### 5.3 D-Sampling

In 2003, Schuster, Wolff, et al. proposed a distributed sampling algorithm called *D-Sampling* [24]. This algorithm is a combination of a centralized sampling algorithm and the DDM algorithm Schuster and Wolff presented in 2001. It assumes a shared-nothing architecture. D-Sampling assumes a centralized dataset and distributes it during runtime. Each node gets the "responsibility" for a set of items. The algorithm loads a sample of the dataset into memory. This sample is then distributed according to the responsibility of the different nodes, fragmenting the dataset vertically. A modified version of the DDM algorithm is then run on the dataset on each node. After a set of possible frequent itemsets is generated, the entire dataset is scanned once to obtain the frequency counts for these candidate itemsets.

Since this algorithm is based on sampling, some frequent itemsets might be missing in the result. In [24] the authors give a tight error bound for this sampling error. D-Sampling is the first distributed frequent itemset mining algorithm assuming a shared-nothing architecture that is not level-based and does not require multiple database scans.

### 5.4 Distributed fp-growth

Zaiane et al. proposed a parallel algorithm that is based on fp-growth in 2001. The algorithm is called *MLFPT* (Multiple Local Frequent Pattern Tree) [27]. It assumes a shared-memory architecture. Just like the centralized fp-growth algorithm, MLFPT does not generate candidates for frequent itemsets but instead builds multiple frequent pattern trees (FP-trees) needing only two full scans of the dataset. Each node is only recording frequent itemsets containing a certain subset of all the items. Thus, each node is responsible for a different set of items.

Since the set of items each node is responsible for is defined after the frequency counts of the 1-itemsets are obtained, the workload can be distributed nearly optimal among the nodes. Unfortunately, this data distribution algorithm is only efficient in a shared-memory architecture and is tailored to the FP-tree data structure. Thus, it can not be used for load balancing in other algorithms.

### 5.5 ZigZag Algorithm

In recent years, research is concentrating on frequent itemset mining in dynamic datasets. The traditional assumption was that the underlying database is static and the data mining task is performed completely anew if changes in the database should be reflected in the data mining results. Opposed to this, mining algorithms on dynamic databases are incremental. They can be seen as algorithms mining a data stream using the landmark window model. The only difference is that in dynamic databases transactions can also be removed, whereas in data streams only new transactions are added.

In [19], Otey et al. propose an algorithm named *ZigZag*. This algorithm assumes a shared-nothing architecture and a setting where the data is initially distributed on different sites (like network data for intrusion detection).

In the ZigZag algorithm, each site first generates the local set of maximal frequent itemsets. Then the global set of maximal frequent itemsets (MFI) can be computed. Using the set MFI, all frequent itemsets can be determined without generating infrequent candidates. The frequency counts of the frequent itemsets can then be computed with a single scan of each local dataset.

Clearly, this approach requires far less communication than the traditional Apriori-based algorithms, since an exchange of the local models does not take place after every round (after generating the $k$-itemsets, the $(k+1)$-itemsets, etc.), but only after the local sets of MFI have been computed.

Additionally, the ZigZag algorithm can be used to determine "high-contrast frequent itemsets" which indicate the skewness of the distributed database. Although the authors do not consider to use this information for dynamic load balancing purposes, the skewness measure can give interesting insight into the properties of the different local datasets.

## 6 Open Problems and Discussion Thereof

### 6.1 Future Work in Distributed Clustering: Things That Could Be Done Better

***Fuzzy Objects*** We wish to represent clusters not only as circles as is generally going to be the case with K-means and other representative choosing cluster representations. The probability function is a great idea that was used, but I'd be curious to explore the idea in a fuller setting. A similar consideration to be made is something as was brought up in the Jim Bosch and Daniel Zinn paper from ECS289f W06. If we can accurately model a cluster based on functions we can perhaps construct a better, more accurate representation without giving away any data points directly. This is not a directly easy idea. When dealing with function approximations of a N-dimensional convex hull, it can become complex. Finding such an approximation, with some adequate statistics could be invaluable to distributed streaming clustering.

***Derived Distribution*** Clustering is something of a unique aspect of data mining in that if you know something about the questions you plan to ask you might be able to reduce a significant amount of the work you plan to do. cosmology can be an illuminating example. Imagine you have received several new data points from a new telescope. You are not necessarily likely to ask what the stars that appear close to it in the sky are like. You are more likely to ask which stars have a similar attributes in a few important columns. This means that even though you may have a spacial correlation you might not be the most likely to use it just yet. therefore you might want to cluster the attributes by any subset of the elements. Because this number grows exponentially with the number of attributes it quickly diminishes the return value of presorting. By use of Amidahl's rule we can decide that the majority of the work will be done in an environment such that the data is distributed arbitrarily from clustering query.

If you are working on a bottom up approach, you distribute the data yourself. You may still want to consider what the attributes you are likely to cluster over are. If you tend to cluster over a very specific three attributes, then it would certainly be worth your while to distribute the data into subspaces of that field. However given the number ($2^n$) of possible subsets one can cluster by, it is somewhat unlikely that we will be able to help too much with the overall clustering.

There is a good possibility of future work to be considered. I plan to grab a couple of datasets from the University of California Irvine repository and run a statistical analysis on them. I'd like to see if there is any reason to distribute the data in a particular pattern as it would be mostly likely to lead to a subspace division for several subsets of the attributes. I hope it should be an interesting, somewhat quick, experiment.

## 6.2 Open Problems in Distributed Frequent Itemset Mining

***Data Distribution*** Only very few algorithms deploy a sophisticated data partitioning strategy for distributing the given dataset. Thus, we see a need for approaches for distributing data specifically for the task of frequent itemset mining. Due to the nature of this data mining task, we believe that distributing the data in a way that suits the deployed mining algorithm will facilitate an improvement in run-time and memory requirements of the overall algorithm. Since most algorithms use a round-robin distribution or an arbitrary partitioning of the entire database, there is much room for future research on specialized data distribution algorithms.

***Dynamic Load Balancing*** Most algorithms assume a homogenous and dedicated environment. But since this assumption might not be valid in many systems, dynamic load balancing techniques should be deployed to distribute the workload equally among the nodes. As we mentioned before, dynamic load balancing is mainly an option for shared-memory architectures.

***Data Stream Processing*** In the case of centralized frequent itemset mining there exists a plethora of algorithms specifically tailored for processing data streams. In the distributed case this development has only started as there are only a few incremental algorithms that process dynamic datasets.

Distributed frequent itemset mining algorithms on data streams are certainly a worthwhile area of research since settings like network traffic monitors for distributed intrusion detection provide a concrete and prominent application. We believe that especially equivalents of centralized sliding window and damped window (where transactions decay over time) approaches are needed for the use in distributed environments. Naturally, when processing data streams in a distributed setting, the notion of approximate mining results using the $\epsilon$ error threshold must also find its translation from the centralized algorithms.

Since data stream processing may be infinite, we propose that in distributed frequent itemset mining algorithms on data streams each node holds a part of a global data structure, just like in the case of the MLFPT algorithm. The global data structure is never computed due to the communication overhead imposed by the rapidly changing local models. Instead, query algorithms are developed that query selected local models in order to determine the requested mining result.

When considering a single high speed data stream, the issue of data distribution becomes even more important than in the case of mining a finite dataset. This is because transactions are constantly arriving and have to be distributed among the available nodes in a certain way in order to achieve nearly equal resource requirements on all different sites. In this scenario, dynamic load balancing is an important topic even in shared-nothing architectures.

## 7 Conclusion

We have shown several algorithms for mining and analyzing distributed data. By giving an overview of these methodologies we can see the current state of the art in distributed data mining.

***Distributed Clustering*** As we can see there are a number of algorithms already available that offer you some choices in clustering distributed data. Depending on what criteria you choose to live by, which subset of the rules you decide to prioritize. If data privacy is of the utmost importance then a algorithm with strong cluster abstraction should be used, such as Kriegal's method. If speed is of the utmost importance, it may be worth ones while to use a relatively simple update of the K-means algorithm. Similar to most everything else in computer science (everywhere perhaps) there are several ways to compute distributed clusters. The methods I have outlined above should give an idea of what some of the primary choices are in the current state of distributed clustering.

In general, I think that density grid based operations are the best. They show the ability to keep the data private while leaving the door open for using a functional abstraction to represent clusters. This methodology can still use some work to make sure that completeness is being generated by the clusters but they seem to be acclimating quickly and are resilient from either a top down or bottom up approach.

***Distributed Frequent Itemset Mining*** Distributed frequent itemset mining is currently not very actively researched. Thus, despite their years of publication, the presented overview papers, like [28, 29], give an up-to-date view on the existing algorithms. Recent papers are focusing on minimizing the communication cost of the prior algorithms and are taking dynamic datasets into account.

Important open problems in the area of distributed frequent itemset mining are twofold. First, in a setting where the dataset is initially in one big database, the crucial question of how to distribute the data to best facilitate the specific data mining task of frequent itemset mining is not solved. Second, equivalents to most centralized frequent itemset mining approaches on data streams are not existing. Especially sliding window approaches and approximate mining algorithms using the common $\epsilon$ error threshold from the centralized case could be useful contributions.

## References

1. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications.* ACM Press New York, NY, USA, 1998.
2. R. Agrawal and J. C. Shafer. Parallel Mining of Association Rules. *IEEE Transactions On Knowledge And Data Engineering*, 8:962–969, 1996.
3. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 1994 International Conference on Very Large Data Bases*, pages 487–499, 1994.
4. Joong Hyuk Chang and Won Suk Lee. A sliding window method for finding recently frequent itemsets over online data streams. *Journal of Information Science and Engineering*, 20(4):753–762, 2004.
5. David Cheung, Jiawei Han, Vincent T. Ng, Ada W. Fu, and Yongjian Fu. A fast distributed algorithm for mining association rules. In *PDIS: International Conference on Parallel and Distributed Information Systems*, 1996.
6. S. Datta, C. Giannella, and H. Kargupta. K-Means Clustering over a Large, Dynamic Network, April 2006.
7. G. Forman and B. Zhang. Distributed Data Clustering Can Be Efficient and Exact. *SIGKDD Explorations*, 2(2):34–38, 2000.
8. Chris Giannella, Jiawei Han, Edward Robertson, and Chao Liu. Mining frequent itemsets over arbitrary time intervals in data streams. Technical report, Indiana University, 2003.
9. A. Gionis, H. Mannila, and P. Tsaparas. Clustering Aggregation. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, Tokyo, Japan, April 2005.
10. Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12, 16-18 Mai 2000.
11. E. Januzaj, H. P. Kriegel, and M. Pfeifle. Scalable Density Based Distributed Clustering. In *Proceedings of EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 88–105, March 2004.
12. E. Januzaj, H.P. Kriegel, and M. Pfeifle. DBDC: Density Based Distributed Clustering. *Proc. 9th Int. Conf. on Extending Database Technology (EDBT 2004), Heraklion, Greece*, pages 88–105, 2004.
13. E. Johnson and H. Kargupta. Collective, Hierarchical Clustering From Distributed, Heterogeneous Data. In M. Zaki and C. Ho, editors, *Large-Scale Parallel KDD Systems. Lecture Notes in Computer Science*, volume 1759, pages 221–244. Springer-Verlag, 1999.
14. M. Klusch, S. Lodi, and G. L. Moro. Distributed Clustering Based on Sampling Local Density Estimates. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 485–490, Mexico, August 2003.
15. H.-P. Kriegel, P. Kröger, A. Pryakhin, and M. Schubert. Effective and Efficient Distributed Model-based Clustering. In *The Fifth IEEE International Conference on Data Mining (ICDM'05)*, Houston, TX, November 2005.

16. Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, August 2002.
17. TK Moon. The expectation-maximization algorithm. *Signal Processing Magazine, IEEE*, 13(6):47–60, 1996.
18. H. Nagesh, S. Goil, and A. Choudhary. Parallel algorithms for clustering high-dimensional large-scale datasets. *Data Mining for Scientific and Engineering Applications*, 2.
19. M.E. Otey, C. Wang, S. Parthasarathy, A. Veloso, and Jr. W. Meira. Mining frequent itemsets in distributed and dynamic databases. In *ICDM 2003: Third IEEE International Conference on Data Mining*, pages 617– 620, Nov. 2003.
20. Byung-Hoon Park and Hillol Kargupta. Distributed data mining: Algorithms, systems, and applications. *Data Mining Handbook*, 2002.
21. Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In *The VLDB Journal*, pages 432–444, 1995.
22. Assaf Schuster and Ran Wolff. Communication-efficient distributed mining of association rules. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 473–484, New York, NY, USA, May 2001. ACM Press.
23. Assaf Schuster, Ran Wolff, and Dan Trock. A high-performance distributed algorithm for mining association rules. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 291, Washington, DC, USA, 2003. IEEE Computer Society.
24. Assaf Schuster, Ran Wolff, and Dan Trock. A high-performance distributed algorithm for mining association rules. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 291, Washington, DC, USA, 2003. IEEE Computer Society.
25. Masahisa Tamura and Masaru Kitsuregawa. Dynamic load balancing for parallel association rule mining on heterogeneous pc cluster systems. In *25th International Conference on Very Large Data Bases*, pages 162–173, 1999.
26. Adriano Veloso, Matthew Eric Otey, Srinivasan Parthasarathy, and Wagner Meira Jr. Parallel and distributed frequent itemset mining on dynamic datasets. pages 184–193, 2003.
27. Osmar R. Zaiane, Mohammad El-Hajj, and Paul Lu. Fast parallel association rule mining without candidacy generation. In *ICDM*, pages 665–668, 2001.
28. Mohammed J. Zaki. Parallel and distributed association mining: A survey. 7(4):14–25, 1999.
29. Mohammed J. Zaki. *Lecture Notes in Computer Science*, volume 1759, chapter Parallel and Distributed Data Mining: An Introduction, pages 1–23. Springer Berlin / Heidelberg, 2000.
30. Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In *Proc. of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 283–286, 1997.