Parallel and Distributed Data Mining: An Introduction

Mohammed J. Zaki

Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
zaki@cs.rpi.edu
http://www.cs.rpi.edu/~zaki

Abstract. The explosive growth in data collection in business and scientific fields has literally forced upon us the need to analyze and mine useful knowledge from it. Data mining refers to the entire process of extracting useful and novel patterns/models from large datasets. Due to the huge size of data and amount of computation involved in data mining, high-performance computing is an essential component for any successful large-scale data mining application. This chapter presents a survey on large-scale parallel and distributed data mining algorithms and systems, serving as an introduction to the rest of this volume. It also discusses the issues and challenges that must be overcome for designing and implementing successful tools for large-scale data mining.

1 Introduction

Data Mining and Knowledge Discovery in Databases (KDD) is a new interdisciplinary field merging ideas from statistics, machine learning, databases, and parallel and distributed computing. It has been engendered by the phenomenal growth of data in all spheres of human endeavor, and the economic and scientific need to extract useful information from the collected data. The key challenge in data mining is the extraction of knowledge and insight from massive databases.

Data mining refers to the overall process of discovering new patterns or building models from a given dataset. There are many steps involved in the KDD enterprise which include data selection, data cleaning and preprocessing, data transformation and reduction, data-mining task and algorithm selection, and finally post-processing and interpretation of discovered knowledge [1,2]. This KDD process tends to be highly iterative and interactive.

Typically data mining has the two high level goals of *prediction* and *description* [1]. In prediction, we are interested in building a model that will predict unknown or future values of attributes of interest, based on known values of some attributes in the database. In KDD applications, the description of the data in human-understandable terms is equally if not more important than prediction. Two main forms of data mining can be identified [3]. In *verification-driven* data mining the user postulates a hypothesis, and the system tries to validate it.

M.J. Zaki, C.-T. Ho (Eds.): Large-Scale Parallel Data Mining, LNAI 1759, pp. 1–23, 2000.

The common verification-driven operations include query and reporting, multidimensional analysis or On-Line Analytical Processing (OLAP), and statistical analysis. *Discovery-driven* mining, on the other hand, automatically extracts new information from data, and forms the main focus of this survey. The typical discovery-driven tasks include association rules, sequential patterns, classification and regression, clustering, similarity search, deviation detection, etc.

While data mining has its roots in the traditional fields of machine learning and statistics, the sheer volume of data today poses the most serious problem. For example, many companies already have data warehouses in the terabyte range (e.g., FedEx, UPS, Walmart). Similarly, scientific data is reaching gigantic proportions (e.g., NASA space missions, Human Genome Project). Traditional methods typically made the assumption that the data is memory resident. This assumption is no longer tenable. Implementation of data mining ideas in high-performance parallel and distributed computing environments is thus becoming crucial for ensuring system scalability and interactivity as data continues to grow inexorably in size and complexity.

Parallel data mining (PDM) deals with tightly-coupled systems including shared-memory systems (SMP), distributed-memory machines (DMM), or clusters of SMP workstations (CLUMPS) with a fast interconnect. Distributed data mining (DDM), on the other hand, deals with loosely-coupled systems such as a cluster over a slow Ethernet local-area network. It also includes geographically distributed sites over a wide-area network like the Internet. The main differences between PDM to DDM are best understood if view DDM as a gradual transition from tightly-coupled, fine-grained parallel machines to loosely-coupled medium-grained LAN of workstations, and finally very coarse-grained WANs. There is in fact a significant overlap between the two areas, especially at the medium-grained level where is it hard to draw a line between them.

In another view, we can think of PDM as an essential component of a DDM architecture. An individual site in DDM can be a supercomputer, a cluster of SMPs, or a single workstation. In other words, each site supports PDM locally. Multiple PDM sites constitute DDM, much like the current trend in meta- or super-computing. Thus the main difference between PDM and DDM is that of scale, communication costs, and data distribution. While, in PDM, SMPs can share the entire database and construct a global mined model, DMMs generally partition the database, but still generate global patterns/models. On the other hand, in DDM, it is typically not feasible to share or communicate data at all; local models are built at each site, and are then merged/combined via various methods.

PDM is the ideal choice in organizations with centralized data-stores, while DDM is essential in cases where there are multiple distributed datasets. In fact, a successful large-scale data mining effort requires a hybrid PDM/DDM approach, where parallel techniques are used to optimize the local mining at a site, and where distributed techniques are then used to construct global or consensus patterns/models, while minimizing the amount of data and results communicated. In this chapter we adopt this unified view of PDM and DDM.

This chapter provides an introduction to parallel and distributed data mining. We begin by explaining the PDM/DDM algorithm design space, and then go on to survey current parallel and distributed algorithms for associations, sequences, classification and clustering, which are the most common mining techniques. We also include a section on recent systems for distributed mining. After reviewing the open challenges in PDM/DDM, we conclude by providing a roadmap for the rest of this volume.

2 Parallel and Distributed Data Mining

Parallel and distributed computing is expected to relieve current mining methods from the sequential bottleneck, providing the ability to scale to massive datasets, and improving the response time. Achieving good performance on today's multiprocessor systems is a non-trivial task. The main challenges include synchronization and communication minimization, work-load balancing, finding good data layout and data decomposition, and disk I/O minimization, which is especially important for data mining.

2.1 Parallel Design Space

The parallel design space spans a number of systems and algorithmic components including the hardware platform, the kind of parallelism exploited, the load balancing strategy, the data layout and the search procedure used.

Distributed Memory Machines vs. Shared Memory Systems. The performance optimization objectives change depending on the underlying architecture. In DMMs synchronization is implicit in message passing, so the goal becomes communication optimization. For shared-memory systems, synchronization happens via locks and barriers, and the goal is to minimize these points. Data decomposition is very important for distributed memory, but not for shared memory. While parallel I/O comes for "free" in DMMs, it can be problematic for SMP machines, which typically serialize I/O. The main challenge for obtaining good performance on DMM is to find a good data decomposition among the nodes, and to minimize communication. For SMP the objectives are to achieve good *data locality*, i.e., maximize accesses to local cache, and to avoid/reduce *false sharing*, i.e., minimize the ping-pong effect where multiple processors may be trying to modify different variables which coincidentally reside on the same cache line. For today's non-uniform memory access (NUMA) hybrid and/or hierarchical machines (e.g., cluster of SMPs), the optimization parameters draw from both the DMM and SMP paradigms.

Another classification of the different architectures comes from the database literature. Here, *shared-everything* refers to the shared-memory paradigm, with a global shared memory and common disks among all the machines. *Shared-nothing* refers to distributed-memory architecture, with a local memory and disk for each processor. A third paradigm called *shared-disks* refers to the mixed case where processors have local memories, but access common disks [4,5].

Task vs. Data Parallelism. These are the two main paradigms for exploiting algorithm parallelism. Data parallelism corresponds to the case where the database is partitioned among P processors. Each processor works on its local partition of the database, but performs the same computation of evaluating candidate patterns/models. Task parallelism corresponds to the case where the processors perform different computations independently, such as evaluating a disjoint set of candidates, but have/need access to the entire database. SMPs have access to the entire data, but for DMMs this can be done via selective replication or explicit communication of the local data. Hybrid parallelism combining both task and data parallelism is also possible, and in fact desirable for exploiting all available parallelism in data mining methods.

Static vs. Dynamic Load Balancing. In static load balancing work is initially partitioned among the processors using some heuristic cost function, and there is no subsequent data or computation movement to correct load imbalances which result from the dynamic nature of mining algorithms. Dynamic load balancing seeks to address this by stealing work from heavily loaded processors and re-assigning it to lightly loaded ones. Computation movement also entails data movement, since the processor responsible for a computational task needs the data associated with that task as well. Dynamic load balancing thus incurs additional costs for work/data movement, but it is beneficial if the load imbalance is large and if load changes with time. Dynamic load balancing is especially important in multi-user environments with transient loads and in heterogeneous platforms, which have different processor and network speeds. These kinds of environments include parallel servers, and heterogeneous, meta-clusters. With very few exceptions, most extant parallel mining algorithms use only a static load balancing approach that is inherent in the initial partitioning of the database among available nodes. This is because they assume a dedicated, homogeneous environment.

Horizontal vs. Vertical Data Layout. The standard input database for mining is a relational table having N rows, also called feature vectors, transactions, or records, and M columns, also called dimensions, features, or attributes. The data layout can be row-wise or column-wise. Many data mining algorithms assume a *horizontal* or row-wise database layout, where they store, as a unit, each transaction (*tid*), along with the attribute values for that transaction. Other methods use a *vertical* or column-wise database layout, where they associate with each attribute a list of all tids (called *tidlist*) containing the item, and the corresponding attribute value in that transaction. Certain mining operations a more efficient using a horizontal format, while others are more efficient using a vertical format.

Complete vs. Heuristic Candidate Generation. The final results of a mining method may be sets, sequences, rules, trees, networks, etc., ranging from simple patterns to more complex models, based on certain search criteria. In the intermediate steps several *candidate* patterns or partial models are evaluated, and the final result contains only the ones that satisfy the (user-specified) input parameters. Mining algorithms can differ in the way new candidates are generated for evaluation. One approach is that of *complete* search, which is guaranteed to generate and test all valid candidates consistent with the data. Note that completeness doesn't mean exhaustive, since pruning can be used to eliminate useless branches in the search space. *Heuristic* generation sacrifices completeness for the sake of speed. At each step, it only examines a limited number (or only one) of "good" branches. *Random* search is also possible. Generally, the more complex the mined model, the more the tendency towards heuristic or greedy search.

Candidate and Data Partitioning. An easy way to discuss the many parallel and distributed mining methods is to describe them in terms of the computation and data partitioning methods used. For example, the database itself can be shared (in shared-memory or shared-disk architectures), partially or totally replicated, or partitioned (using round-robin, hash, or range scheduling) among the available nodes (in distributed-memory architectures).

Similarly, the candidate concepts generated and evaluated in the different mining methods can be shared, replicated or partitioned. If they are shared then all processors evaluate a single copy of the candidate set. In the replicated approach the candidate concepts are replicated on each machine, and are first evaluated locally, before global results are obtained by merging them. Finally, in the partitioned approach, each processor generates and tests a disjoint candidate concept set.

In the sections below we describe parallel and distributed algorithms for some of the typical discovery-driven mining tasks including associations, sequences, decision tree classification and clustering. Table 1 summarizes in list form where each parallel algorithm for each of the above mining tasks lies in the design space. It would help the reader to refer to the table while reading the algorithm descriptions below.

2.2 Association Rules

Given a database of transactions, where each transaction consists of a set of items, association discovery finds all the item sets that frequently occur together, the so called *frequent itemsets*, and also the rules among them. An example of an association could be that, "40% of people who buy Jane Austen's *Pride and Prejudice* also buy *Sense and Sensibility.*" Potential application areas include catalog design, store layout, customer segmentation, telecommunication alarm diagnosis, etc.

The Apriori [6] method serves as the base algorithm for the vast majority of parallel association algorithms. Apriori uses a complete, bottom-up search, with a horizontal data layout and enumerates all frequent itemsets. Apriori is an iterative algorithm that counts itemsets of a specific length in a given database pass. The process starts by scanning all transactions in the database and computing the frequent items. Next, a set of potentially frequent *candidate* itemsets

Algorithm	Base Algorithm	Machine	Parallelism	LoadBal	DB Layout	Concepts	Database
Association Rule Mining							
CD, PEAR, PDM, FDM, NPA	Apriori	DMM	Data	Static	Horizontal	Replicated	Partitioned
DD, SPA, IDD	Apriori	DMM	Task	Static	Horizontal	Partitioned	Partitioned
HD	Apriori	DMM	Hybrid	Hybrid	Horizontal	Hybrid	Partitioned
CCPD	Apriori	SMP	Data	Static	Horizontal	Shared	Partitioned
CandD, HPA, HPA-ELD	Apriori	DMM	Task	Static	Horizontal	Partitioned	Partially Replicated
PCCD	Apriori	SMP	Task	Static	Horizontal	Partitioned	Shared
APM	DIC	SMP	Task	Static	Horizontal	Shared	Partitioned
PPAR	Partition	DMM	Task	Static	Horizontal	Replicated	Partitioned
PE, PME, PC, PMC	Eclat, Clique	CLUMPS	Task	Static	Vertical	Partitioned	Partially Replicated
Sequence Mining							
NPSPM	GSP	DMM	Data	Static	Horizontal	Replicated	Partitioned
SPSPM	GSP	DMM	Task	Static	Horizontal	Partitioned	Partitioned
HPSPM	GSP	DMM	Task	Static	Horizontal	Partitioned	Partially Replicated
pSPADE	SPADE	SMP	Task	Dynamic	Vertical	Partitioned	Shared
D-MSDD	MSDD	DMM	Task	Static	Horizontal	Partitioned	Replicated
Decision Tree Classification							
SPRINT, SLIQ/R, SLIQ/D, ScalParC	SLIQ/SPRINT	DMM	Data	Static	Vertical	Replicated	Partitioned
DP-att, DP-rec, PDT	C4.5	DMM	Data	Static	Horizontal	Replicated	Partitioned
MWK	SPRINT	SMP	Data	Dynamic	Vertical	Shared	Shared
SUBTREE	SPRINT	SMP	Hybrid	Dynamic	Vertical	Partitioned	Partitioned
HTF	SPRINT	DMM	Hybrid	Dynamic	Vertical	Partitioned	Partitioned
pCLOUDS	CLOUDS	DMM	Hybrid	Dynamic	Horizontal	Partitioned	Partitioned
Clustering							
P-CLUSTER	K-Means	DMM	Data	Static	Horizontal	Replicated	Partitioned
MAFIA	-	DMM	Task	Static	Horizontal	Partitioned	Partitioned

Table 1. Design Space for Parallel Mining Algorithms: Associations, Sequences, Classification and Clustering.

of length 2 is formed from the frequent items. Another database scan is made to obtain their supports. The frequent itemsets are retained for the next pass, and the process is repeated until all frequent itemsets (of various lengths) have been enumerated.

Other sequential methods for associations that have been parallelized, include DHP [7], which tries to reduce the number of candidates by collecting approximate counts (using hash tables) in the previous level. These counts can be used to rule out many candidates in the current pass that cannot possibly be frequent. The *Partition* algorithm [8] minimizes I/O by scanning the database only twice. It partitions the database into small chunks which can be handled in memory. In the first pass it generates a set of all potentially frequent itemsets, and in the second pass it counts their global frequency. In both phases it uses a vertical database layout. The DIC algorithm [9] dynamically counts candidates of varying length as the database scan progresses, and thus is able to reduce the number of scans.

A completely different design characterizes the equivalence class based algorithms (Eclat, MaxEclat, Clique, and MaxClique) proposed by Zaki et al. [10]. These methods utilize a vertical database format, complete search, a mix of bottom-up and hybrid search, and generate a mix of maximal and non-maximal frequent itemsets. The algorithms utilize the structural properties of frequent itemsets to facilitate fast discovery. The items are organized in a subset lattice search space, which is decomposed into small independent chunks or sub-lattices, which can be solved in memory. Efficient lattice traversal techniques are used, which quickly identify all the frequent itemsets via tidlist intersections.

Replicated or Shared Candidates, Partitioned Database. The candidate concepts in association mining are the frequent itemsets. A common paradigm for parallel association mining is to partition the database in equal-sized horizontal blocks, with the candidate itemsets replicated on all processors. For Apriori-based parallel methods, in each iteration, each processor computes the frequency of the candidate set in its local database partition. This is followed by a sum-reduction to obtain the global frequency. The infrequent itemsets are discarded, while the frequent ones are used to generate the candidates for the next iteration.

Barring minor differences, the methods that follow this data-parallel approach include PEAR [11], PDM [12], Count Distribution (CD) [13], FDM [14], Non-Partitioned Apriori (NPA) [15], and CCPD [16]. CCPD uses shared-memory machines, and thus maintains a shared candidate set among all processors. It also parallelizes the candidate generation.

The other algorithms use distributed-memory machines. PDM, based on DHP, prunes candidates using approximate counts from the previous level. It also does parallelizes candidate generation, at the cost of an extra round of communication. The remaining methods simply replicate the computation for candidate generation. FDM is further optimized to work on distributed sites. It uses novel pruning techniques to minimize the number of candidates, and thus the communication during sum-reduction.

The advantage of replicated candidates and partitioned database, for Apriori-based methods, is that they incur only a small amount of communication. In each iteration only the frequencies of candidate concepts are exchanged; no data is exchanged. These methods thus outperform the pure partitioned candidates approach described in the next section. Their disadvantage is that the aggregate system memory is not used effectively, since the candidates are replicated.

Other parallel algorithms, that use a different base sequential method include APM [17], a task-parallel, shared-memory, asynchronous algorithm, based on DIC. Each processor independently applies DIC to its local partition. The candidate set is shared among processors, but is updated asynchronously when a processor inserts new itemsets.

PPAR [11], a task-parallel, distributed-memory algorithm, is built upon Partition, with the exception that PPAR uses the horizontal data format. Each processor gathers the locally frequent itemsets of all sizes in one pass over their local database (which may be partitioned into chunks as well). All potentially frequent itemsets are then broadcast to other processors. Then each processor gathers the counts of these global candidates in the second local pass. Finally a broadcast is performed to obtain the globally frequent itemsets.

Partitioned Candidates, Partitioned Database. Algorithms implementing this approach include Data Distribution (DD) [13], Simply-Partitioned Apriori (SPA) [15], and Intelligent Data Distribution (IDD) [18]. All three are Apriori-based, and employ task parallelism on distributed-memory machines. Here each processor computes the frequency of a disjoint set of candidates. However, to find the global support each processor must scan the entire database, both its local partition, and other processor's partitions (which are exchanged in each iteration). The main advantage of these methods is that they utilize the aggregate system-wide memory by evaluating disjoint candidates, but they are impractical for any realistic large-scale dataset.

The Hybrid Distribution (HD) algorithm [18] adopts a middle ground between Data Distribution and Count Distribution. It utilizes the aggregate memory, and also minimizes communication. It partitions the P processors into G equal-sized groups. Each of the G groups is considered a super-processor, and applies Count Distribution, while the P/G processors within a group use Intelligent Data Distribution. The database is horizontally partitioned among the G super-processors, and the candidates are partitioned among the P/G processors in a group. HD cuts down the database communication costs by $1/G$.

Partitioned Candidates, Selectively Replicated or Shared Database. A third approach is to evaluate a disjoint candidate set and to selectively replicate the database on each processor. Each processor has all the information to generate and test candidates asynchronously. Methods in this paradigm are Candidate Distribution (CandD) [13], Hash Partitioned Apriori (HPA) [15], HPA-ELD [15], and PCCD [16], all of which are Apriori-based. PCCD uses SMP machines, and accesses a shared-database, but is not competitive with CCPD. Candidate Distribution is also outperformed by Count Distribution. Nevertheless, HPA-ELD, a hybrid between HPA and NPA, was shown to be better than NPA, SPA, and HPA.

Zaki et al. [19] proposed four algorithms, ParEclat (PE), ParMaxEclat (PME), ParClique (PC), and ParMaxClique (PMC), targeting hierarchical systems like clusters of SMP machines. The data is assumed to be vertically partitioned among the SMP machines. After an initial tidlist exchange phase and class scheduling phase, the algorithms proceed asynchronously. In the asynchronous phase each processor has available the classes assigned to it, and the tidlists for all items. Thus each processor can independently generate all frequent itemsets from its classes. No communication or synchronization is required. Further, all available memory of the system is used, no in-memory hash trees are needed, and only simple intersection operations are required for itemset enumeration.

Most of the extant association mining methods use a static load balancing scheme; a dynamic load balancing approach on a heterogeneous cluster has been presented in [20]. For more detailed surveys of parallel and distributed association mining see [21] and the chapter by Joshi et al. in this volume.

2.3 Sequential Patterns

Sequence discovery aims at extracting frequent events that commonly occur over a period of time [22]. An example of a sequential pattern could be that “70% of the people who buy Jane Austen’s *Pride and Prejudice* also buy *Emma* within a month”. Sequential pattern mining deals with purely categorical domains, as opposed to the real-valued domains used in time-series analysis. Examples of categorical domains include text, DNA, market baskets, etc.

In essence, sequence mining is “temporal” association mining. However, while association rules discover only intra-transaction patterns (itemsets), we now also have to discover inter-transaction patterns (sequences) across related transactions. The set of all frequent sequences is an superset of the set of frequent itemsets. Hence, sequence search is much more complex and challenging than itemset search, thereby necessitating fast parallel algorithms.

Serial algorithms for sequence mining that have been parallelized include GSP [23], MSDD [24], and SPADE [25]. GSP is designed after Apriori. It computes the frequency of candidate sequences of length k in iteration k. The candidates are generated from the frequent sequences from the previous iteration. MSDD discovers patterns in multiple event sequences; it explores the rule space directly instead of the sequence space. SPADE is similar to Eclat. It uses vertical layout and temporal joins to compute frequency. The search space is broken into small memory-resident chunks, which are explored in depth- or breadth-first manner.

Three parallel algorithms based on GSP were presented in [26]. All three methods use the partitioned database approach, and are distributed-memory based. NPSPM (with replicated candidates) is equivalent to NPA, SPSPM (with partitioned candidates) the same as SPA and HPSPM is equivalent to HPA, which have been described above. HPSPM performed the best among the three. A parallel and distributed implementation of MSDD was presented in [27].

A shared-memory, SPADE-based parallel algorithm, utilizing dynamic load balancing is described by Zaki, and new algorithms for parallel sequence mining are also described by Joshi et al. in this volume.

2.4 Classification

Classification aims to assign a new data item to one of several predefined categorical classes [28,29]. Since the field being predicted is pre-labeled, classification is also known as supervised induction. While there are several classification methods including neural networks [30] and genetic algorithms [31], decision trees [32,33] are particularly suited to data mining, since they can be constructed relatively quickly, and are simple and easy to understand. Common applications of classification include credit card fraud detection, insurance risk analysis, bank loan approval, etc.

A decision tree is built using a recursive partitioning approach. Each internal node in the tree represents a decision on an attribute, which splits the database into two or more children. Initially the root contains the entire database, with examples from mixed classes. The split point chosen is the one that best separates or discriminates the classes. Each new node is recursively split in the same manner until a node contains only one or a majority class.

Decision tree classifiers typically use a greedy search over the space of all possible trees; there are simply too many trees to allow a complete search. The search is also biased towards simple trees. Existing classifiers have used both the horizontal and vertical database layouts. In parallel decision tree construction the candidate concepts are the possible split points for all attributes within a node of the expanding tree. For numeric attributes a split point is of the form $A \leq v_i$, and for categorical attributes the test takes the form $A \in \{v_1, v_2, ...\}$, where v_i is a value from the domain of attribute A.

Below we look at some parallel decision tree methods. Recent surveys on parallel and scalable induction methods are also presented in [34,35].

Replicated Tree, Partitioned Database. SLIQ [36] was one of the earliest scalable decision tree classifiers. It uses a vertical data format, called attribute lists, allowing it to pre-sort numeric attributes in the beginning, thus avoiding the repeated sorting required at each node in traditional tree induction. Nevertheless it uses a memory-resident structure called *class-list*, which grows linearly in the number of input records. SPRINT [37] removes this memory dependence, by storing the classes as part of the attribute lists. It uses data parallelism, and a distributed-memory platform.

In SPRINT and parallel versions of SLIQ, the attribute lists are horizontally partitioned among all processors. The decision tree is also replicated on all processors. The tree is constructed synchronously in a breadth-first manner. Each processor computes the best split point, using its local attribute lists, for all the nodes on the current tree level. A round of communication takes place to determine the best split point among all processors. Each processor independently splits the current nodes into new children using the best split point, setting the stage for the next tree level. Since a horizontal record is split in multiple attribute lists, a hash table is used to note which record belongs to which child.

The parallelization of SLIQ follows a similar paradigm, except for the way the class list is treated. SLIQ/R uses a replicated class list, while SLIQ/D uses a distributed class list. Experiments showed that while SLIQ/D is better able to exploit available memory, SLIQ/R was better in terms of performance, but SPRINT outperformed both SLIQ/R and SLIQ/D.

ScalParC [38] is also an attribute-list-based parallel classifier for distributed-memory machines. It is similar in design to SLIQ/D (except that it uses hash tables per node, instead of global class lists). It uses a novel distributed hash table for splitting a node, reducing the communication complexity and memory requirements over SPRINT, making it scalable to larger datasets.

The DP-rec and DP-att [39] algorithms exploit record-based and attribute-based data parallelism, respectively. In record-based data parallelism (also used in SPRINT, ScalParC SLIQ/D and SLIQ/R), the records or attribute lists are horizontally partitioned among the processors. In contrast, in attribute-based data parallelism, the attributes are divided so that each processor is responsible for an equal number of attributes. In both the schemes processors cooperate to expand a tree node. Local computations are performed in parallel, followed by information exchanges to get a global best split point.

Parallel Decision Tree (PDT) [40], a distributed-memory, data-parallel algorithm, splits the training records horizontally in equal-sized blocks, among the processors. It follows a master-slave paradigm, where the master builds the tree, and finds the best split points. The slaves are responsible for sending class frequency statistics to the master. For categorical attributes, each processor gathers local class frequencies, and forwards them to the master. For numeric attributes, each processor sorts the local values, finds class frequencies for split points, and exchanges these with all other slaves. Each slave can then calculate the best local split point, which is sent to the master, who then selects the best global split point.

Shared Tree, Shared Database. MWK (and its precursors BASIC and FWK) [41], a shared-memory implementation based on SPRINT uses this approach. MWK uses dynamic attribute-based data parallelism. Multiple processors co-operate to build a shared decision tree in a breadth-first manner. Using a dynamic scheduling scheme, each processor acquires an attribute for any tree node at the current level, and evaluates the split points, before processing another attribute. The processor that evaluates the last attribute of a tree node, also computes the best split point for that node. Similarly, the attribute lists are split among the children using attribute parallelism.

Hybrid Tree Parallelism. SUBTREE [41] uses dynamic task parallelism (that exists in different sub-trees) combined with data parallelism on shared-memory systems. Initially all processors belong to one group, and apply data parallelism at the root. Once new child nodes are formed, the processors are also partitioned into groups, so that a group of child nodes can be processed in parallel by a processor group. If the tree nodes associated with a processor group become pure (i.e., contain examples from a single class), then these processors join some other active group.

The Hybrid Tree Formulation (HTF) in [42] is very similar to SUBTREE. HTF uses distributed memory machines, and thus data redistribution is required in HTF when assigning a set of nodes to a processor group, so that the processor group has all records relevant to an assigned node.

pCLOUDS [43] is a distributed-memory parallelization of CLOUDS [44]. It does not require attribute lists or the pre-sorting for numeric attributes; instead it samples the split points for numeric attributes followed by an estimation step to narrow the search space for the best split. It thus reduces both computation and I/O requirements. pCLOUDS employs a mixed parallelism approach. Initially, data parallelism is applied for nodes with many records. All small nodes are queued to be processed later using task parallelism. Before processing small nodes the data is redistributed so that all required data is available locally at a processor.

2.5 Clustering

Clustering is used to partition database records into subsets or clusters, such that elements of a cluster share a set of common properties that distinguish them from other clusters [45,46,47,48]. The goal is to maximize intra-cluster and minimize inter-cluster similarity. Unlike classification which has predefined labels, clustering must in essence automatically come up with the labels. For this reason clustering is also called unsupervised induction. Applications of clustering include demographic or market segmentation for identifying common traits of groups of people, discovering new types of stars in datasets of stellar objects, and so on.

The K-means algorithm is a popular clustering method. The idea is to randomly pick K data points as cluster centers. Next, each record or point is assigned to the cluster it is closest to in terms of squared-error or Euclidean distance. A new center is computed by taking the mean of all points in a cluster, setting the stage for the next iteration. The process stops when the cluster centers cease to change. Parallelization of K-means received a lot of attention in the past. Different parallel methods, mainly using hypercube computers, appear in [49,50,51,52]. We do not describe these methods in detail, since they used only small memory-resident datasets.

Hierarchical clustering represents another common paradigm. These methods start with a set of distinct points, each forming its own cluster. Then recursively, two clusters that are close are merged into one, until all points belong to a single cluster. In [49,53], parallel hierarchical agglomerative clustering algorithms were presented, using several inter-cluster distance metrics and parallel computer architectures. These methods also report results on small datasets.

P-CLUSTER [54] is a distributed-memory client-server K-means algorithm. Data is partitioned into blocks on a server, which sends initial cluster centers and data blocks to each client. A client assigns each record in its local block to the nearest cluster, and sends results back to the server. The server then recalculates the new centers and another iteration begins. To further improve performance P-CLUSTER uses that the fact that after the first few iterations only a few records change cluster assignments, and also the centers have less tendency to move in later iterations. They take advantage of these facts to reduce the number of distance calculations, and thus the time of the clustering algorithm.

Among the recent methods, MAFIA [55], is a distributed memory algorithm for subspace clustering. Traditional methods, like K-means and hierarchical clustering, find clusters in the whole data space, i.e., they use all dimensions for distance computations. Subspace clustering focuses on finding clusters embedded in subsets of a high-dimensional space. MAFIA uses adaptive grids (or bins) in each dimension, which are merged to find clusters in higher dimensions. Parallel implementation of MAFIA is similar to association mining. The candidates here are the potentially dense units (the subspace clusters) in k dimensions, which have to be tested if they are truly dense. MAFIA employs task parallelism, where data as well as candidates are equally partitioned among all processors. Each processor computes local density, followed by a reduction to obtain global density.

The paper by Dhillon and Modha in this volume presents a distributed-memory parallelization of K-means, while the paper by Johnson and Kargupta describes a distributed hierarchical clustering method.

2.6 Distributed Mining Frameworks

Recently, there has been an increasing interest in distributed and wide-area data mining systems. The fact that many global businesses and scientific endeavors require access to multiple, distributed, and often heterogeneous databases, underscores the growing importance of distributed data mining.

An ideal platform for DDM is a cluster of machines at a local site, or cluster of clusters spanning a wide area, the so-called computational grids, connected via Internet or other high speed networks. As we noted earlier, PDM is best viewed as a local component within a DDM system. Further the main differences between the two is the cost of communication or data movement, and the fact that DDM must typically handle multiple (possibly heterogeneous) databases. Below we review some recent efforts in developing DDM frameworks.

Most methods/systems for DDM assume that the data is horizontally partitioned among the sites, and is homogeneous (share the same feature space). Each site mines its local data and generates locally valid concepts. These concepts are exchanged among all the sites to obtain the globally valid concepts. The Partition [8] algorithm for association mining is a good example. It is inherently suitable for DDM. Each site can generate locally frequent itemsets at a given threshold level. All local results are combined and then evaluated at each site to obtain the globally frequent itemsets.

Another example is JAM [56,57], a java-based multi-agent system utilizing meta-learning, used primarily in fraud-detection applications. Each agent builds a classification model, and different agents are allowed to build classifiers using different techniques. JAM also provides a set of meta-learning agents for combining multiple models learnt at different sites into a *meta-classifier* that in many cases improves the overall predictive accuracy. Knowledge Probing [58] is another approach to meta-learning. Knowledge probing retains a descriptive model after combining multiple classifiers, rather than treating the meta-classifier as a black-box. The idea is to learn on a separate dataset, the class predictions from all the local classifiers.

PADMA [59] is an agent based architecture for distributed mining. Individual agents are responsible for local data access, hierarchical clustering in text document classification, and web based information visualization. The BODHI [60] DDM system is based on the novel concept of collective data mining. Naive mining of heterogeneous, vertically partitioned, sites can lead to an incorrect global data model. BODHI guarantees correct local and global analysis with minimum communication.

In [61] a new distributed do-all primitive, called D-DOALL, was described that allows easy scheduling of independent mining tasks on a network of workstations. The framework allows incremental reporting of results, and seeks to reduce communication via resource-aware task scheduling principles.

The Papyrus [62] java-based system specifically targets wide-area DDM over clusters and meta-clusters. It supports different data, task and model strategies. For example, it can move models, intermediate results or raw data between nodes. It can support coordinated or independent mining, and various methods for combining local models. Papyrus uses PMML (Predictive Model Markup Language) to describe and exchange mined models. Kensignton [63] is another java-based system for distributed enterprise data mining. It is a three-tiered system, with a client front-end for GUI, and visual programming of data mining tasks. The middle-layer application server provides persistent storage, task execution control, and data management and preprocessing functions. The third-tier implements a parallel data mining service.

Other recent work in DDM includes decision tree construction over distributed databases [64], where the learning agents can only exchange summaries instead of raw data, and the databases may have shared attributes. The main challenge is to construct a decision tree using implicit records rather than materializing a join over all the datasets. The WoRLD system [65] describes an inductive rule-learning program that learns from data distributed over a network. WoRLD also avoids joining databases to create a central dataset. Instead it uses marker-propagation to compute statistics. A marker is a label of a class of interest. Counts of the different markers are maintained with each attribute value, and used for evaluating rules. Markers are propagated among different tables to facilitate distributed learning.

For more information on parallel and distributed data mining see the book by Freitas and Lavington [66] and the edited volume by Kargupta and Chan [67]. Also see [68] for a discussion of cost-effective measures for assessing the performance of a mining algorithm before implementing it.

3 Research Issues and Challenges

In this section we highlight some of the outstanding research issues and a number of open problems for designing and implementing the next-generation large-scale mining methods and KDD systems.

High Dimensionality. Current methods are only able to hand a few thousand dimensions or attributes. Consider association rule mining as an example. The second iteration of the algorithm counts the frequency of all pairs of items, which has quadratic complexity. In general, the complexity of different mining algorithms may not be linear in the number of dimensions, and new parallel methods are needed that are able to handle large number of attributes.

Large Size. Databases continue to increase in size. Current methods are able to (perhaps) handle data in the gigabyte range, but are not suitable for terabyte-sized data. Even a single scan for these databases is considered expensive. Most current algorithms are iterative, and scan data multiple times. For example, it is an open problem to mine all frequent associations in a single pass, although sampling based methods show promise [69,70]. In general, minimizing the number of data scans is paramount. Another factor limiting the scalability of most mining algorithms is that they rely on in-memory data structures for storing potential patterns and information about them (such as candidate hash tree [6] in associations, tid hash table [71] in classification). For large databases these structures will certainly not fit in aggregate system memory. This means that temporary results will have to be written out to disk or the database will have to be divided into partitions small enough to be processed in memory, entailing further data scans.

Data Location. Today's large-scale data sets are usually logically and physically distributed, requiring a decentralized approach to mining. The database may be horizontally partitioned where different sites have different transactions, or it may be vertically partitioned, with different sites having different attributes. Most current work has only dealt with the horizontal partitioning approach. The databases may also have heterogeneous schemas.

Data Type. To-date most data mining research has focused on structured data, as it is the simplest, and most amenable to mining. However, support for other data types is crucial. Examples include unstructured or semi-structured (hyper)text, temporal, spatial and multimedia databases. Mining these is fraught with challenges, but is necessary as multimedia content and digital libraries proliferate at astounding rates. Techniques from parallel and distributed computing will lie at the heart of any proposed scalable solutions.

Data Skew. One of the problems adversely affecting load balancing in parallel mining algorithms is sensitivity to data skew. Most methods partition the database horizontally in equal-sized blocks. However, the number of patterns generated from each block can be heavily skewed, i.e., while one block may contribute many, the other may have very few patterns, implying that the processor responsible for the latter block will be idle most of the time. Randomizing the blocks is one solution, but it is still not adequate, given the dynamic and interactive nature of mining. The effect of skewness on different algorithms needs to be further studied (see [72] for some recent work).

Dynamic Load Balancing. Most extant algorithms use only a static partitioning scheme based on the initial data decomposition, and they assume a homogeneous, dedicated environment. This is far from reality. A typical parallel database server has multiple users, and has transient loads. This calls for an investigation of dynamic load balancing schemes. Dynamic load balancing is also crucial in a heterogeneous environment, which can be composed of meta- and super-clusters, with machines ranging from ordinary workstations to supercomputers.

Incremental Methods. Everyday new data is being collected, and existing data stores are being updated with the new data or purged of the old one. To-date there have been no parallel or distributed algorithms that are incremental in nature, which can handle updates and deletions without having to recompute patterns or rules over the entire database.

Multi-table Mining, Data Layout, and Indexing Schemes. Almost no work has been done on mining over multiple tables or over distributed databases which have different schemas. Data in a warehouse is typically arranged in a star schema, with a central fact table (e.g., point-of-sales data), and associated dimension tables (e.g., product information, manufacturer, etc.). Traditional mining over these multiple tables would first require us to create a large single table that is the join of all the tables. The joined table also has tremendous amounts of redundancy. We need better methods for processing such multiple tables, without having to materialize a single large view. Also, little work has been done on the optimal or near-optimal data layout or indexing schemes for fast data access for mining.

Parallel DBMS/File Systems. To-date most results reported have hand-partitioned the database, mainly horizontally, on different processors. There has been very little study conducted in using a parallel database/file system for managing the partitioned database, and the accompanying striping, and layout issues. Recently there has been increasing emphasis on tight database integration of mining [73,74,75,76], but it has mainly been confined to sequential approaches. Some exceptions include Data Surveyor [77], a mining tool that uses the Monet database server for parallel classification rule induction. Also, generic set-oriented primitive operations were proposed in [78] for classification and clustering. These primitives were fully integrated with a parallel DBMS.

Interaction, Pattern Management, and Meta-level Mining. The KDD process is highly interactive, as the human participates in almost all the steps. For example, the user is heavily involved in the initial data understanding, selection, cleaning, and transformation phases. These steps in fact consume more time than mining *per se*. Moreover, depending on the parameters of the search, mining methods may generate too many patterns to be analyzed directly. One needs methods to allow meta-level queries [79,80,81] on the results, to impose constraints that focus on patterns of interest [82,83], to refine or generalize rules [84,85], etc. Thus there is a need for a complete set of tools that query and mine the pattern/model database as well. Parallel methods can be successful in providing the desired rapid response in all of the above steps.

4 Book Organization

This book contains chapters covering all the major tasks in data mining including parallel and distributed mining frameworks, associations, sequences, clustering and classification. We provide a brief synopsis of each chapter below, organized under four main headings.

4.1 Mining Frameworks

Graham Williams et al. present Data Miner's Arcade, a java-based platform-independent system for integrating multiple analysis and mining tools, using a common API, and providing seamless data access across multiple systems. Components of the DM Arcade include parallel algorithms (e.g., BMARS - multiple adaptive regression B-splines), virtual environments for data visualization, and data management for mining.

Bailey et al. describe the implementation of Osiris, a data server for wide-area distributed data mining, built upon clusters, meta-clusters (with commodity network like Internet) and super-clusters (with high-speed network). Osiris addresses three key issues: What data layout should be used on the server? What tradeoffs are there in moving data or predictive models between nodes? How data should be moved to minimize latency; what protocols should be used? Experiments were performed on a wide-area system linking Chicago and Washington via the NSF/MCI vBNS network.

Parthasarathy et al. present InterAct, an active mining framework for distributed mining. Active mining refers to methods that maintain valid mined patterns or models in the presence of user interaction and database updates. The framework uses mining summary structures that are maintained across updates or changes in user specifications. InterAct also allows effective client-server data and computation sharing. Active mining results were presented on a number of methods like discretization, associations, sequences, and similarity search.

4.2 Association Rules and Sequences

Joshi et al. open this section with a survey chapter on parallel mining of association rules and sequences. They discuss the many extant parallel solutions, and give an account of the challenges and issues for effective formulations of discovering frequent itemsets and sequences.

Morishita and Nakaya describe a novel parallel algorithm for mining correlated association rules. They mine rules based on the chi-squared metric that optimizes the statistical significance or correlation between the rule antecedent and consequent. A parallel branch-and-bound algorithm was proposed that uses a term rewriting technique to avoid explicitly maintaining lists of open and closed nodes on each processor. Experiments on SMP platforms (with up to 128 processors) show very good speedups.

Shintani and Kitsuregawa propose new load balancing strategies for generalized association rule mining using a gigabyte-sized database on a cluster of 100 PCs connected with an ATM network. In generalized associations the items are at the leaf levels in a hierarchy or taxonomy of items, and the goal is to discover rules involving concepts at multiple (and mixed) levels. They show that load balancing is crucial for performance on such large-scale clusters.

Zaki presents pSPADE, a parallel algorithm for sequence mining. pSPADE divides the pattern search space into disjoint, independent sub-problems based on suffix-classes, each of which can be solved in parallel in an asynchronous manner. Task parallelism and dynamic inter- and intra-class load balancing is used for good performance. Results on a 12 processor SMP using up to a 1 GB dataset show good speedup and scaleup.

4.3 Classification

Skillicorn presents parallel techniques for generating predictors for classification and regression models. A recent trend in learning is to build multiple prediction models on different samples from the training set, and combine them, allowing faster induction and lower error rates. This framework is highly amenable to parallelism and forms the focus of this paper.

Goil and Choudhary implemented a parallel decision tree classifier using the aggregates computed in multidimensional analysis or OLAP. They compute aggregates/counts per class along various dimensions, which can then be used for computing the attribute split-points. Communication is minimized by coalescing messages and is done once per tree level. Experiments on a 16 node IBM SP2 were presented.

Hall et al. describe distributed rule induction for learning a single model from disjoint datasets. They first learn local rules from a single site; these are merged to form a global rule set. They show that while this approach promises fast induction, accuracy tapers off (as compared to directly mining the whole database) as the number of sites increases. They suggested some heuristics to minimize this loss in accuracy.

4.4 Clustering

Johnson and Kargupta present the Collective Hierarchical Clustering algorithm for clustering over distributed, heterogeneous databases. Rather than gathering the data at a central site, they generate local cluster models, which are subsequently combined to obtain the global clustering.

Dhillon and Modha parallelized the K-means clustering algorithm on a 16 node IBM SP2 distributed-memory system. They exploit the inherent data parallelism of the K-means algorithm, by performing the point-to-centroid distance calculations in parallel. They demonstrated linear speedup on a 2GB dataset.

5 Conclusion

We conclude by observing that the need for large-scale data mining algorithms and systems is real and immediate. Parallel and distributed computing is essential for providing scalable, incremental and interactive mining solutions. The field is in its infancy, and offers many interesting research directions to pursue. We hope that this volume, representing the state-of-the-art in parallel and distributed mining methods, will be successful in bringing to surface the requirement and challenges in large-scale parallel KDD systems.

References

1. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From data mining to knowledge discovery: An overview. [86]
2. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: The KDD process for extracting useful knowledge from volumes of data. Communications of the ACM **39** (1996)
3. Simoudis, E.: Reality check for data mining. IEEE Expert: Intelligent Systems and Their Applications **11** (1996) 26–33
4. DeWitt, D., Gray, J.: Parallel database systems: The future of high-performance database systems. Communications of the ACM **35** (1992) 85–98
5. Valduriez, P.: Parallel database systems: Open problems and new issues. Distributed and Parallel Databases **1** (1993) 137–165
6. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In Fayyad, U., et al, eds.: Advances in Knowledge Discovery and Data Mining, AAAI Press, Menlo Park, CA (1996) 307–328
7. Park, J.S., Chen, M., Yu, P.S.: An effective hash based algorithm for mining association rules. In: ACM SIGMOD Intl. Conf. Management of Data. (1995)
8. Savasere, A., Omiecinski, E., Navathe, S.: An efficient algorithm for mining association rules in large databases. In: 21st VLDB Conf. (1995)
9. Brin, S., Motwani, R., Ullman, J., Tsur, S.: Dynamic itemset counting and implication rules for market basket data. In: ACM SIGMOD Conf. Management of Data. (1997)
10. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New algorithms for fast discovery of association rules. In: 3rd Intl. Conf. on Knowledge Discovery and Data Mining. (1997)

11. Mueller, A.: Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, University of Maryland, College Park (1995)
12. Park, J.S., Chen, M., Yu, P.S.: Efficient parallel data mining for association rules. In: ACM Intl. Conf. Information and Knowledge Management. (1995)
13. Agrawal, R., Shafer, J.: Parallel mining of association rules. IEEE Trans. on Knowledge and Data Engg. **8** (1996) 962–969
14. Cheung, D., Han, J., Ng, V., Fu, A., Fu, Y.: A fast distributed algorithm for mining association rules. In: 4th Intl. Conf. Parallel and Distributed Info. Systems. (1996)
15. Shintani, T., Kitsuregawa, M.: Hash based parallel algorithms for mining association rules. In: 4th Intl. Conf. Parallel and Distributed Info. Systems. (1996)
16. Zaki, M.J., Ogihara, M., Parthasarathy, S., Li, W.: Parallel data mining for association rules on shared-memory multi-processors. In: Supercomputing'96. (1996)
17. Cheung, D., Hu, K., Xia, S.: Asynchronous parallel algorithm for mining association rules on shared-memory multi-processors. In: 10th ACM Symp. Parallel Algorithms and Architectures. (1998)
18. Han, E.H., Karypis, G., Kumar, V.: Scalable parallel data mining for association rules. In: ACM SIGMOD Conf. Management of Data. (1997)
19. Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: Parallel algorithms for fast discovery of association rules. Data Mining and Knowledge Discovery: An International Journal **1(4):343-373** (1997)
20. Tamura, M., Kitsuregawa, M.: Dynamic load balancing for parallel association rule mining on heterogeneous PC cluster systems. In: 25th Intl Conf. on Very Large Data Bases. (1999)
21. Zaki, M.J.: Parallel and distributed association mining: A survey. IEEE Concurrency **7** (1999) 14–25
22. Agrawal, R., Srikant, R.: Mining sequential patterns. In: 11th Intl. Conf. on Data Engg. (1995)
23. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: 5th Intl. Conf. Extending Database Technology. (1996)
24. Oates, T., Schmill, M.D., Jensen, D., Cohen, P.R.: A family of algorithms for finding temporal structure in data. In: 6th Intl. Workshop on AI and Statistics. (1997)
25. Zaki, M.J.: Efficient enumeration of frequent sequences. In: 7th Intl. Conf. on Information and Knowledge Management. (1998)
26. Shintani, T., Kitsuregawa, M.: Mining algorithms for sequential patterns in parallel: Hash based approach. In: 2nd Pacific-Asia Conf. on Knowledge Discovery and Data Mining. (1998)
27. Oates, T., Schmill, M.D., Cohen, P.R.: Parallel and distributed search for structure in multivariate time series. In: 9th European Conference on Machine Learning. (1997)
28. Weiss, S.M., Kulikowski, C.A.: Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. Morgan Kaufman (1991)
29. Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine Learning, Neural and Statistical Classification. Ellis Horwood (1994)
30. Lippmann, R.: An introduction to computing with neural nets. IEEE ASSP Magazine **4** (1987)
31. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Morgan Kaufmann (1989)

32. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth, Belmont (1984)
33. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufman (1993)
34. Provost, F., Aronis, J.: Scaling up inductive learning with massive parallelism. Machine Learning **23** (1996)
35. Provost, F., Kolluri, V.: A survey of methods for scaling up inductive algorithms. Data Mining and Knowledge Discovery: An International Journal **3** (1999) 131–169
36. Mehta, M., Agrawal, R., Rissanen, J.: SLIQ: A fast scalable classifier for data mining. In: Proc. of the Fifth Intl Conference on Extending Database Technology (EDBT), Avignon, France (1996)
37. Shafer, J., Agrawal, R., Mehta, M.: Sprint: A scalable parallel classifier for data mining. In: 22nd VLDB Conference. (1996)
38. Joshi, M., Karypis, G., Kumar, V.: ScalParC: A scalable and parallel classification algorithm for mining large datasets. In: Intl. Parallel Processing Symposium. (1998)
39. Chattratichat, J., Darlington, J., Ghanem, M., Guo, Y., Huning, H., Kohler, M., Sutiwaraphun, J., To, H.W., Dan, Y.: Large scale data mining: Challenges and responses. In: 3rd Intl. Conf. on Knowledge Discovery and Data Mining. (1997)
40. Kufrin, R.: Decision trees on parallel processors. In Geller, J., Kitano, H., Suttner, C., eds.: Parallel Processing for Artificial Intelligence 3, Elsevier-Science (1997)
41. Zaki, M.J., Ho, C.T., Agrawal, R.: Parallel classification for data mining on shared-memory multiprocessors. In: 15th IEEE Intl. Conf. on Data Engineering. (1999)
42. Srivastava, A., Han, E.H., Kumar, V., Singh, V.: Parallel formulations of decision-tree classification algorithms. Data Mining and Knowledge Discovery: An International Journal **3** (1999) 237–261
43. Sreenivas, M., Alsabti, K., Ranka, S.: Parallel out-of-core divide and conquer techniques with application to classification trees. In: 13th International Parallel Processing Symposium. (1999)
44. Alsabti, K., Ranka, S., Singh, V.: Clouds: A decision tree classifier for large datasets. In: 4th Intl Conference on Knowledge Discovery and Data Mining. (1998)
45. Jain, A.K., Dubes, R.C.: Algorithms for Clustering Data. Prentice Hall (1988)
46. Cheeseman, P., Kelly, J., Self, M., et al.: AutoClass: A Bayesian classification system. In: 5th Intl Conference on Machine Learning, Morgan Kaufman (1988)
47. Fisher, D.H.: Knowledge acquisition via incremental conceptual clustering. Machine Learning **2** (1987)
48. Michalski, R.S., Stepp, R.E.: Learning from observation: Conceptual clustering. In Michalski, R.S., Carbonell, J.G., Mitchell, T.M., eds.: Machine Learning: An Artificial Intelligence Approach. Volume I. Morgan Kaufmann (1983) 331–363
49. Li, X., Fang, Z.: Parallel clustering algorithms. Parallel Computing **11** (1989) 270–290
50. Rivera, F., Ismail, M., Zapata, E.: Parallel squared error clustering on hypercube arrays. Journal of Parallel and Distributed Computing **8** (1990) 292–299
51. Ranka, S., Sahni, S.: Clustering on a hypercube multicomputer. IEEE Trans. on Parallel and Distributed Systems **2(2)** (1991) 129–137
52. Rudolph, G.: Parallel clustering on a unidirectional ring. In et al., R.G., ed.: Transputer Applications and Systems '93: Volume 1. IOS Press, Amsterdam (1993) 487–493
53. Olson, C.: Parallel algorithms for hierarchical clustering. Parallel Computing **21** (1995) 1313–1325
54. Judd, D., McKinley, P., Jain, A.: Large-scale parallel data clustering. In: Intl Conf. Pattern Recognition. (1996)

55. S. Goil, H.N., Choudhary, A.: MAFIA: Efficient and scalable subspace clustering for very large data sets. Technical Report 9906-010, Center for Parallel and Distributed Computing, Northwestern University (1999)
56. Stolfo, S., Prodromidis, A., Tselepis, S., Lee, W., Fan, W., Chan, P.: Jam: Java agents for meta-learning over distributed databases. In: 3rd Intl. Conf. on Knowledge Discovery and Data Mining. (1997)
57. Prodromidis, A., Stolfo, S., Chan, P.: Meta-learning in distributed data mining systems: Issues and approaches. [67]
58. Guo, Y., Sutiwaraphun, J.: Knowledge probing in distributed data mining. In: 3rd Pacific-Asia Conference on Knowledge Discovery and Data Mining. (1999)
59. Kargupta, H., Hamzaoglu, I., Stafford, B.: Scalable, distributed data mining using an agent based architecture. In: 3rd Intl. Conf. on Knowledge Discovery and Data Mining. (1997)
60. Kargupta, H., Park, B.H., Hershberger, D., Johnson, E.: Collective data mining: A new perspective toward distributed data mining. [67]
61. Parthasarathy, S., Subramonian, R.: Facilitating data mining on a network of workstations. [67]
62. Grossman, R.L., Bailey, S.M., Sivakumar, H., Turinsky, A.L.: Papyrus: A system for data mining over local and wide area clusters and super-clusters. In: Supercomputing'99. (1999)
63. Chattratichat, J., Darlington, J., Guo, Y., Hedvall, S., Kohler, M., Syed, J.: An architecture for distributed enterprise data mining. In: 7th Intl. Conf. High-Performance Computing and Networking. (1999)
64. Bhatnagar, R., Srinivasan, S.: Pattern discovery in distributed databases. In: AAAI National Conference on Artificial Intelligence. (1997)
65. Aronis, J., Kolluri, V., Provost, F., Buchanan, B.: The WoRLD: Knowledge discovery from multiple distributed databases. In: Florida Artificial Intelligence Research Symposium. (1997)
66. Freitas, A., Lavington, S.: Mining very large databases with parallel processing. Kluwer Academic Pub., Boston, MA (1998)
67. Kargupta, H., Chan, P., eds.: Advances in Distributed Data Mining. AAAI Press, Menlo Park, CA (2000)
68. Skillicorn, D.: Strategies for parallel data mining. IEEE Concurrency **7** (1999) 26–35
69. Toivonen, H.: Sampling large databases for association rules. In: 22nd VLDB Conf. (1996)
70. Zaki, M.J., Parthasarathy, S., Li, W., Ogihara, M.: Evaluation of sampling for data mining of association rules. In: 7th Intl. Wkshp. Research Issues in Data Engg. (1997)
71. Shafer, J., Agrawal, R., Mehta, M.: SPRINT: A scalable parallel classifier for data mining. In: Proc. of the 22nd Intl Conference on Very Large Databases, Bombay, India (1996)
72. Cheung, D., Xiao, Y.: Effect of data distribution in parallel mining of associations. Data Mining and Knowledge Discovery: An International Journal **3** (1999) 291–314
73. Agrawal, R., Shim, K.: Developing tightly-coupled data mining applications on a relational database system. In: 2nd Intl. Conf. on Knowledge Discovery in Databases and Data Mining. (1996)
74. Meo, R., Psaila, G., Ceri, S.: A new SQL-like operator for mining association rules. In: 22nd Intl. Conf. Very Large Databases. (1996)
75. Meo, R., Psaila, G., Ceri, S.: A tightly-coupled architecture for data mining. In: Intl. Conf. on Data Engineering. (1998)

76. Sarawagi, S., Thomas, S., Agrawal, R.: Integrating association rule mining with databases: alternatives and implications. In: ACM SIGMOD Intl. Conf. Management of Data. (1998)
77. Holsheimer, M., Kersten, M.L., Siebes, A.: Data surveyor: Searching the nuggets in parallel. [86]
78. Lavington, S., Dewhurst, N., Wilkins, E., Freitas, A.: Interfacing knowledge discovery algorithms to large databases management systems. Information and Software Technology **41** (1999) 605–617
79. Kamber, M., Han, J., Chiang, J.Y.: Metarule-guided mining of multi-dimensional association rules using data cubes. In: 3rd Intl. Conf. on Knowledge Discovery and Data Mining. (1997)
80. Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H., Verkamo, A.I.: Finding interesting rules from large sets of discovered association rules. In: 3rd Intl. Conf. Information and Knowledge Management. (1994) 401–407
81. Shen, W.M., Ong, K.L., Mitbander, B., Zaniolo, C.: Metaqueries for data mining. [86]
82. Ng, R.T., Lakshmanan, L., Jan, J., Pang, A.: Exploratory mining and pruning optimizations of constrained association rules. In: ACM SIGMOD Intl. Conf. Management of Data. (1998)
83. Srikant, R., Vu, Q., Agrawal, R.: Mining Association Rules with Item Constraints. In: 3rd Intl. Conf. on Knowledge Discovery and Data Mining. (1997)
84. Matheus, C., Piatetsky-Shapiro, G., McNeill, D.: Selecting and reporting what is interesting. In Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., eds.: Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press (1996)
85. Toivonen, H., Klemettinen, M., Ronkainen, P., Hätönen, K., Mannila, H.: Pruning and grouping discovered association rules. In: MLnet Wkshp. on Statistics, Machine Learning, and Discovery in Databases. (1995)
86. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., eds.: Advances in Knowledge Discovery and Data Mining. AAAI Press, Menlo Park, CA (1996)