# 3rd IPDPS Workshop on High Performance Data Mining

## Preface

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 workshop provided a forum for presenting recent results in high performance computing for data mining including applications, algorithms, software, and systems. High-performance was broadly interpreted to include scalable sequential as well as parallel and distributed algorithms and systems. Relevant topics for the workshop included:

1. Scalable and/or parallel/distributed algorithms for various mining tasks like classification, clustering, sequences, associations, trend and deviation detection, etc.
2. Methods for pre/post-processing like feature extraction and selection, discretization, rule pruning, model scoring, etc.
3. Frameworks for KDD systems, and parallel or distributed mining.
4. Integration issues with databases and data-warehouses.

These proceedings contain 9 papers that were accepted for presentation at the workshop. Each paper was reviewed by two members of the program committee. In keeping with the spirit of the workshop some of these papers also represent work-in-progress. In all cases, however, the workshop program highlights avenues of active research in high performance data mining.

We would like to thank all the authors and attendees for contributing to the success of the workshop. Special thanks are due to the program committee and external reviewers for help in reviewing the submissions.

February 2000

Mohammed J. Zaki
Vipin Kumar
David B. Skillicorn
Editors

## Workshop Co-Chairs

Mohammed J. Zaki (Rensselaer Polytechnic Institute, USA)
Vipin Kumar (University of Minnesota, USA)
David B. Skillicorn (Queens University, Canada)

## Program Committee

Philip K. Chan (Florida Institute of Technology, USA)
Alok Choudhary (Northwestern University, USA)
Umeshwar Dayal (Hewlett-Packard Labs., USA)
Alex A. Freitas (Pontifical Catholic University of Parana, Brazil)
Ananth Grama (Purdue University, USA)
Robert Grossman (University of Illinois-Chicago, USA)
Yike Guo (Imperial College, UK)
Jiawei Han (Simon Fraser University, Canada)
Howard Ho (IBM Almaden Research Center, USA)
Chandrika Kamath (Lawrence Livermore National Labs., USA)
Masaru Kitsuregawa (University of Tokyo, Japan)
Sanjay Ranka (University of Florida, USA)
Vineet Singh (Hewlett-Packard Labs., USA)
Domenico Talia (ISI-CNR: Institute of Systems Analysis and Information Technology, Italy)
Kathryn Burn-Thornton (Durham University, UK)

## External Reviewers

Eui-Hong (Sam) Han (University of Minnesota, USA)
Wen Jin (Simon Fraser University, Canada)
Harsha S. Nagesh (Northwestern University, USA)
Srinivasan Parthasarathy (University of Rochester, USA)

# Implementation Issues in the Design of I/O Intensive Data Mining Applications on Clusters of Workstations

R. Baraglia[1], D. Laforenza[1], Salvatore Orlando[2],
P. Palmerini[1] and Raffaele Perego[1]

[1] Istituto CNUCE, Consiglio Nazionale delle Ricerche (CNR), Pisa, Italy
[2] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy

**Abstract** This paper investigates *scalable* implementations of out-of-core I/O-intensive Data Mining algorithms on affordable parallel architectures, such as clusters of workstations. In order to validate our approach, the K-means algorithm, a well known *DM Clustering* algorithm, was used as a test case.

## 1 Introduction

Data Mining (DM) applications exploit huge amounts of data, stored in files or databases. Such data need to be accessed to discover patterns and correlations useful for various purposes, above all for guiding strategic decision making in the business domain. Many DM applications are strongly I/O intensive since they need to read and process the input dataset several times [1,6,7]. Several techniques have been proposed in order to improve the performance of DM applications. Many of them are based on parallel processing [5]. In general, their main goals are to reduce the computation time and/or reduce the time spent on accessing out-of-memory data.

Since the early 1990s there has been an increasing trend to move away from expensive and specialized proprietary parallel supercomputers towards clusters of workstations (COWs) [3]. Historically, COWs have been used primarily for science and engineering applications, but their low cost, scalability, and generality provide a wide array of opportunities for new domains of application [13]. DM is certainly one of these domains, since DM algorithms generally exhibit large amounts of data parallelism. However, to efficiently exploit COWs, parallel implementations should be adaptive with respect to the specific features of the machine (e.g. they must take into account the memory hierarchies and caching policies adopted by modern hardware/software architectures).

Specific *Out-of-Core* (OoC) techniques (also known as External Memory techniques) [3,14] can be exploited to approach DM problems that require huge amounts of memory. OoC techniques are useful for all applications that do not completely fit into the physical memory. Their main goal is to reduce memory hierarchy overheads by bypassing the OS virtual memory system and explicitly

managing I/O. Direct control over data movements between main memory and secondary storage is achieved by splitting the dataset into several small blocks. These blocks are then loaded into data structures which will certainly fit into physical memory. They are processed and, if necessary, written back to disks. The knowledge of the patterns used by the algorithm to access the data can be exploited in an effective way to reduce I/O overheads by overlapping them with useful computations. The access pattern exploited by the DM algorithm discussed in this paper is simple, since read-only datasets are accessed sequentially and iteratively. Note that the general purpose external memory mechanism provided by the operating system – in our case, the Unix `read()` system call – is specifically optimized for this kind of data access.

This paper investigates *scalable* implementations of I/O-intensive DM algorithms on affordable parallel architectures, such as clusters of PCs equipped with main memories of a limited size, which are not sufficiently big to store the whole dataset (or even a partition of it). The test case DM application used to validate our approach is based on the on-line K-means algorithm, a well known *DM Clustering* algorithm [8,10]. The testbed COW was composed of three SMPs, interconnected by a 100BaseT switched Ethernet, where each SMP was equipped with two Pentium II - 233 MHz processors, 128 MB of main memory, and a 4GB UW-SCSI disk. Their OS was Linux, kernel version 2.2.5-15. The paper is organized as follows. Section 2 discusses implementation issues related to the design of I/O-intensive DM applications. Section 3 deals with the K-means algorithm and its parallel implementation. Finally, Section 4 discusses the results of our experiments and draws some conclusions.

## 2 Implementation of I/O Intensive DM Applications

As mentioned above, we are interested in DM algorithms that access sequentially the same dataset several times. The repeated scanning of the whole dataset entails good spatial locality but scarce temporal locality. The latter can only be exploited if the whole dataset entirely fits into the physical memory. In general, however, this condition cannot be taken for granted because "real life" datasets are generally very large. Moreover, the physical memory is of limited size, and other running processes contend for its usage. The adoption of an OoC algorithm, which takes advantage of possible prefetching policies implemented by both software drivers and disk controllers [11], and which allows to exploit *multitasking* or *multithreading* strategies in order to overlap I/O latencies with useful computations, is thus mandatory.

The best policy might thus appear to be to adopt OoC algorithms only if a dataset does not fit into the physical memory. When the memory is large enough, an in-core approach might seem more efficient, since all the dataset is read once from disk, and is repeatedly accessed without further I/O operations. Clearly such an in-core strategy might fail when other processes use the main memory, thus causing swapping on the disk. We believe that "smart" OoC approaches are always preferable to their in-core counterparts, even when datasets are small

with respect to memory size. This assertion is due to the existence of a *buffer cache* for block devices in modern OSs, such as Linux [2]. The available physical memory left unused by the kernel and processes is dynamically enrolled in the buffer cache on demand. When the requirement for primary memory increases, for example because new processes enter the system, the memory allocated to buffers is reduced. We conducted experiments to compare in-core and out-of-core versions of a simple test program that repeatedly scans a dataset which fits into physical memory. We observed that the two versions of the program have similar performances. In fact, if we consider the OoC version of this simple program, at the end of the first scan the buffer cache contains the blocks of the whole dataset. The following scans of the dataset will not actually access the disk at all, since they find all the blocks to be read in the main memory, i.e. in the buffer cache. In other words, due to the mechanisms provided by the OS, the actual behavior of the OoC program becomes in-core.
We also observed another advantage of the OoC program over the in-core solution. During the first scan of the dataset, the OoC program takes advantage of OS prefetching. In fact, during the processing of a block the OS prefetches the next one, thus hiding some I/O time. On the contrary, I/O time of in-core programs cannot be overlapped with useful computations because the whole dataset has to be read before starting the computation.

In summary, the OoC approach not only works well for small datasets, but it also scales-up when the problem size exceeds the physical memory size, i.e., in those cases when in-core algorithms fail due to memory swapping. Moreover, to improve scalability for large datasets, we can also exploit multitasking techniques in conjunction with OoC techniques to hide I/O time. To exploit multitasking, non-overlapping partitions of the whole dataset must be assigned to distinct tasks. The same technique can also be used to parallelize the application, by mapping these tasks onto distinct machines. This kind of data-parallel paradigm is usually very effective for implementing DM algorithms, since computation is generally uniform, data exchange between tasks is limited, and generally involves a global synchronization at the end of each scan of the whole dataset. This synchronization is used to check termination conditions and to restore a consistent global state. Consistency restoration is needed since the tasks start each iteration on the basis of a consistent state, generating new local states that only reflect their partial view of the whole dataset.
Finally, parallel DM algorithms implemented on COWs also have to deal with load imbalance. In fact, workload imbalance may derive either from different capacities of the machines involved or from unexpected arrivals of external jobs. Since the programming paradigm adopted is data parallel, a possible solution to this problem is to dynamically change partition sizes.

## 3 A Test Case DM Algorithm and its Implementation

There is a variety of applications, ranging from marketing to biology, astrophysics, and so on [8], that need to identify subsets of records (clusters) present-

ing characteristics of *homogeneity*. In this paper we used a well known clustering algorithm, the **K-means** algorithm [10] as a case study representative of a class of I/O intensive DM algorithms. We deal with the on-line formulation of K-means, which can be considered as a competitive learning formulation of the classical K-means algorithm. K-means considers records in a dataset to be represented as *data-points* in a high dimensional space. Clusters are identified by using the concept of proximity among data-points in this space. The K-means algorithm is known to have some limitations regarding the dependence on the initial conditions and the shape and size of the clusters found [9,10]. Moreover, it is necessary to define *a priori* the number $K$ of clusters that we expect to find, even though it is also possible to start with a small number of clusters (and associated centers), and increase this number when specific conditions are observed. The three main steps of the on-line K-means sequential algorithm are: (1) start with a given number of centers randomly chosen; (2) scan all the data-points of the dataset, and for each point $p$ find the center closest to $p$, assign $p$ to the cluster associated with this center, and move the center toward $p$; (3) repeat step 2 until the assignment of data-points to the various clusters remains unchanged. Note that the repetition of step 2 ensures that centers gradually get attracted into the middle of the clusters. In our tests we used synthetic datasets and we fixed a priori $K$.

*Parallel Implementation.* We implemented the OoC version of the algorithm mentioned above, where data-points are repeatedly scanned by sequentially reading small blocks of 4 KBytes from the disk. The program was implemented using MPI according to an SPMD paradigm. A non overlapping partition of the input file, univocally identified by a pair of boundaries, is processed by each task of the SPMD program. The number of tasks involved in the execution may be greater than the number of physical processors, thus exploiting multitasking. This parallel formulation of our test case is similar to those described in [12,4], and requires a new consistent global state to be established once each scan of the whole dataset is completed. Our global state corresponds with the new positions reached by the $K$ centers. These positions are determined by summing the vectors corresponding with the centers' movements which were separately computed by the various tasks involved. In our implementation, the new center positions are computed by a single task, the root one, and are broadcast to the others. The root task also checks the termination condition.

The load balancing strategy adopted is simple but effective. It is based on past knowledge of the bandwidths of all concurrent tasks (i.e. number of points computed in a unit of time). If a load imbalance is detected, the size of the partitions is increased for "faster" tasks and decreased for "slower" ones. This requires input datasets to be replicated on all the disks of our testbed. If complete replication is too expensive or not possible, file partitions with overlapping boundaries can be exploited as well. Let $NP$ be the total number of data-points, and $\{p_1, \ldots, p_n\}$ the $n$ tasks of the SPMD program. At the first iteration $np_i^1 = NP/n$ data-points are assigned to each $p_i$. During iteration $j$ each $p_i$ measures the elapsed time $T_i^j$ spent on elaborating its own block of $np_i^j$ points,

so that $t_i^j = T_i^j / np_i^j$ is the time taken by each $p_i$ to elaborate a single point, and $b_i^j = 1/t_i^j = np_i^j / T_i^j$ is its bandwidth. In order to balance the workload, the numbers $np_i^{j+1}$ of data-points which each $p_i$ has to process in the next iteration are then computed on the basis of the various $b_i^j$ ($np_i^{j+1} = \rho_i^j \cdot NP$, where $\rho_i^j = b_i^j / \sum_{i=1}^{n} b_i^j$). Finally, values $np_i^{j+1}$ are easily translated into partition boundaries, i.e. a pair of offsets within the replicated or partially replicated input file.

## 4 Experimental Results and Conclusions

Several experiments were conducted on the testbed with our parallel implementation of $K$-means based on MPI. Data-parallelism, OoC techniques, multitasking, and load balancing strategies were exploited. Note that the successful adoption of multitasking mainly depends on (1) the number of disks with respect to the number of processors available on each machine, and (2) the computation granularity (i.e., the time spent on processing each data block) with respect to the I/O bandwidth. In our experiments on synthetic datasets, we tuned this computational granularity by changing the number $K$ of clusters to look for. Another important characteristics of our approach is the size of the partitions assigned to the tasks mapped on a single SMP machine. If the sum of these sizes is less than the size of the physical main memory, we guess that the behavior of the OoC application will be similar to its in-core counterpart, due to a large enough buffer cache. Otherwise, sequential accesses carried out by a task to its dataset partition will entail disk accesses, so that the only possibility of hiding these I/O times is to exploit, besides OS prefetching, some form of moderated multitasking.

Figure 1 shows the effects of the presence of the buffer cache. On a single SMP we ran our test case algorithm with a small dataset (64 MB) and small computational granularity ($K$=3). Bars show the time spent by the tasks in computing (`t_comp`), in doing I/O and being idle in some OS queue (`t_io` + `t_idle`), and in communication and synchronization (`t_comm`). The two bars on the left hand side represent the first and the second iterations of a sequential implementation of the test case. The four bars on the right hand side regard the parallel implementation (2 tasks mapped on the same SMP). Note that in both cases the `t_io` and `t_idle` are high during the first iteration, since the buffer cache is not able to fulfill the read requests (cache misses). On the other hand, these times almost disappear from the the second iteration bars, since the accessed blocks are found in the buffer cache (cache hits).

Figure 2 shows the effects of multitasking on a single SMP when the disk has to be accessed. Although a small dataset was used for these experiments, the bars only refer to the first iteration, during which we certainly need to access the disk. Now recall that our testbed machines are equipped with a single disk each. This represents a strong constraint on the I/O bandwidth of our platform. This is particularly evident when several I/O-bound tasks, running in parallel on an SMP, try to access this single disk. In this regard, we found that our test case has different behaviors depending on the computational granularity.

For a fine granularity ($K$=8), the computation is completely I/O-bound. In this condition it is better to allocate a single task to each SMP (see Figure 2.(a)). When we allocated more then one task, the performance worsened because of the limited I/O bandwidth and I/O conflicts. For a coarser granularity ($K = 32$), the performance improved when two tasks were used (see Figure 2.(b)). In the case of higher degrees of parallelism the performance decreases. This is due to the overloading of the single disk, and to noises introduced by multitasking into the OS prefetching policy.

Figure 3 shows some speedup curves. The plots refer to 20 iterations with $K$=16. We used at most two tasks per SMP. Note the super-linear speedup achieved when 2 or 3 processors were used. These processors belong to distinct SMPs, so that this super-linear speedup is due to the exploitation of multiple disks and to the effects of the buffer cache. In fact, when moderately large datasets were used (64 MB or 128 MB) the data partitions associated with the tasks mapped on each SMP fit into the buffer caches. Overheads due to communications, occurring at the end of each iteration, are very small and do not affect speedup.
In the case of a larger dataset (384 MB), whose size is greater than the whole main memory available, when the number of tasks remains under three, linear speedups were obtained. For larger degrees of parallelism, the speedup decreases. This is still due to the limited I/O bandwidth on each SMP.

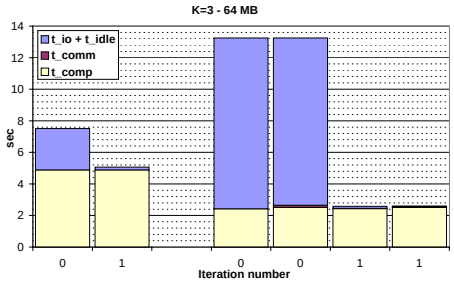



**Figure1.** Execution times of two iterations of the test case on a single SMP.

Figure 4 shows the effectiveness of the load balancing strategy adopted. Both plots refer to experiments conducted using all the six processors of our testbed with the 64MB dataset and $K = 32$. The plot in the left hand side of the figure shows the number of blocks dynamically assigned to each task by our load balancing algorithm as a function of the iteration index. During time interval $[t1, t3]$ ($[t2, t4]$) we executed a CPU-intensive process on the SMP `A` (`M`) running tasks `A0` and `A1` (`M1` and `M1`). As it can be seen, the load balancing algorithm quickly detects the variation in the capacities of the machines, and correspondingly adjusts the size of the partitions by narrowing partitions assigned to slower

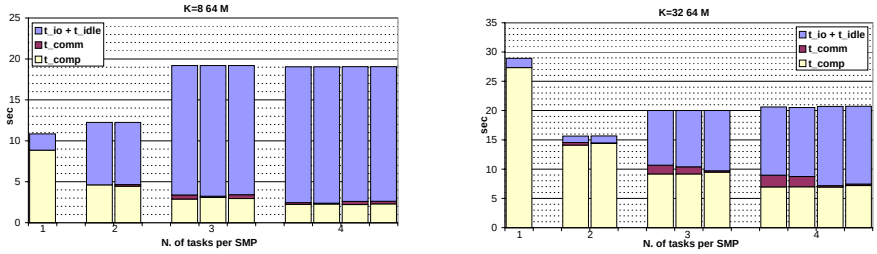



**Figure2.** Execution times of the first iteration on a single SMP by varying the number of tasks exploited and the computational granularities: (a) $K$=8 and (b) $K$=32.

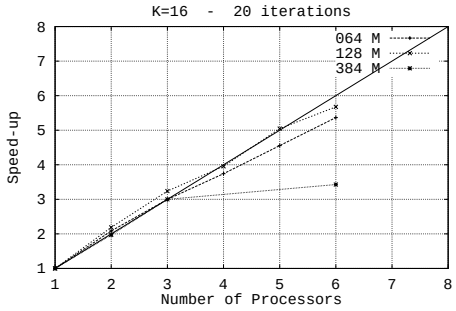



**Figure3.** Speedup curves for different dataset sizes (20 iterations, $K = 16$).

machines and enlarging the others. The plot in the right hand side compares the execution times obtained exploiting or not our load balancing strategy as a function of the external load present in one of the machines. We can see that in the absence of external load the overhead introduced by the load balancing strategy is negligible. As the external load increases, the benefits of exploiting the load balancing strategy increase as well.

In conclusion, this work has investigated the issues related to the implementation of a test case application, chosen as a representative of a large class of DM I/O-intensive applications, on an inexpensive COW. Effective strategies for managing I/O requests and for overlapping their latencies with useful computations have been devised and implemented. Issues related to data parallelism exploitation, OoC techniques, multitasking, and load balancing strategies have been discussed. To validate our approach we conducted several experiments and discussed the encouraging results achieved. Future work regards the evaluation of the possible advantages of exploiting lightweight threads for intra-SMP parallelism and multitasking. Moreover, other I/O intensive DM algorithms have to be considered in order to define a framework of techniques/functionalities useful for efficiently solving general DM applications on COWs, which, unlike homo-

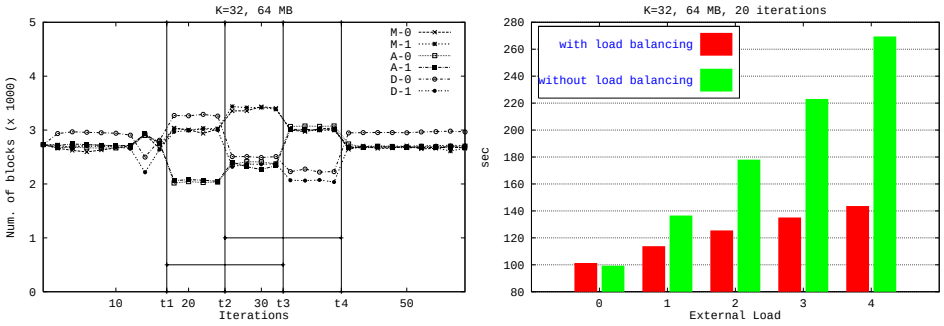



**Figure4.** Effectiveness of the load balancing strategy.

geneous MPPs, impose additional issues that must be addressed using adaptive strategies.

## References


1. Jain A.K. and Dubes R.C. *Algorithms for Clustering Data.* Prentice Hall, 1988.
2. M. Beck et al. *Linux Kernel Internals, 2nd ed.* Addison-Wesley, 1998.
3. Rajkumar Buyya, editor. *High Performance Cluster Computing.* Prentice Hall PTR, 1999.
4. I. S. Dhillon and D. S. Modha. A data clustering algorithm on distributed memory machines. In *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, 1999.
5. A. A. Freitas and S. H. Lavington. *Mining Very Large Databases with Parallel Processing.* Kluwer Academin Publishers, 1998.
6. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining Very Large Databases. *IEEE Computer*, 32(8):38–45, 1999.
7. E. Han, G. Karypis, and V. Kumar. Scalable Parallel Data Mining for Association Rules. *IEEE Transactions on Knowledge and Data Engineering.* To appear.
8. J.A. Hartigan. *Clustering Algorithms.* Wiley & Sons, 1975.
9. G. Karypis, E. Han, and V. Kumar. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *IEEE Computer*, 32:68–75, 1999.
10. Mac Queen, J.B. Some Methods for Classification and Analysis of Multivariate Observation. *$5^{th}$ Berkeley Symp. on Mathematical Statistics and Probability*, pages 281–297. Univ. of California Press, 1967.
11. Chris Ruemmler and John Wilkes. An Introduction to Disk Drive Modeling. *IEEE Computer*, 27(3):17–28, March 1994.
12. K. Stoffel and A. Belkoniene. Parallel k-means clustering for large datasets. *EuroPar'99 Parallel Processing*, Lecture Notes in Computer Science, No. 1685. Springer-Verlag, 1999.
13. Sterling T.L., Salmon J., Becker D.J., and Savarese D.F. *How to Build a Beowulf. A guide to the Implementation and Application of PC Clusters.* The MIT Press, 1999.
14. J. S. Vitter. External Memory Algorithms and Data Structures. In *External Memory Algorithms (DIMACS Series on Discrete Mathematics and Theoretical Computer Science).* American Mathematical Society, 1999.

# A Requirements Analysis for Parallel KDD Systems

William A. Maniatty[1] and Mohammed J. Zaki[2]

[1] Computer Science Dept., University at Albany, Albany, NY 12222
maniatty@cs.albany.edu, http://www.cs.albany.edu/~maniatty/
[2] Computer Science Dept., Rensselaer Polytechnic Institute, Troy, NY 12180
zaki@cs.rpi.edu, http://www.cs.rpi.edu/~zaki/

**Abstract.** The current generation of data mining tools have limited capacity and performance, since these tools tend to be sequential. This paper explores a migration path out of this bottleneck by considering an integrated hardware and software approach to parallelize data mining. Our analysis shows that parallel data mining solutions require the following components: parallel data mining algorithms, parallel and distributed data bases, parallel file systems, parallel I/O, tertiary storage, management of online data, support for heterogeneous data representations, security, quality of service and pricing metrics. State of the art technology in these areas is surveyed with an eye towards an integration strategy leading to a complete solution.

## 1 Introduction

Knowledge discovery in databases (KDD) employs a variety of techniques, collectively called *data mining*, to uncover trends in large volumes of data. Many applications generate (or acquire) data faster than it can be analyzed using existing KDD tools, leading to perpetual data archival without retrieval or analysis. Furthermore, analyzing sufficiently large data sets can exceed the available computational resources of existing computers. In order to reverse the vicious cycle induced by these two problematic trends, the issues of performing KDD faster than the rate of arrival and increasing capacity must simultaneously be dealt with. Fortunately, novel applications of parallel computing techniques should assist in solving these large problems in a timely fashion.

Parallel KDD (PKDD) techniques are not currently that common, though recent algorithmic advances seek to address these problems (Freitas and Lavington 1998; Zaki 1999; Zaki and Ho 2000; Kargupta and Chan 2000). However, there has been no work in designing and implementing large-scale parallel KDD systems, which must not only support the mining algorithms, but also the entire KDD process, including the pre-processing and post-processing steps (in fact, it has been posited that around 80% of the KDD effort is spent in these steps, rather than mining). The picture gets even more complicated when one considers persistent data management of mined patterns and models.

Given the infancy of KDD in general, and PKDD in particular, it is not clear how or where to start, to realize the goal of building a PKDD system

that can handle terabyte-sized (or larger) central or distributed datasets. Part of the problem stems from the fact that PKDD draws input from diverse areas that have been traditionally studied in isolation. Typically, the KDD process is supported by a hierarchical architecture consisting of the following layers: (from bottom to top) I/O Support, File System, Data Base, Query Manager, and Data Mining. However, the current incarnations of this architecture tend to be sequential, limiting both problem size and performance. To implement a successful PKDD toolkit, we need to borrow, adapt, and enhance research in fields such as super-, meta- and heterogeneous-computing environments, parallel and distributed databases, parallel and distributed file systems, parallel I/O, mass storage systems, and so on (not to mention the other fields that make up KDD — statistics, machine learning, visualization, etc.).

This paper represents a first step in the process of unifying these diverse technologies and leveraging them within the PKDD system. We do this by discussing the system requirements for PKDD and the extant solutions (or lack thereof), i.e., the *what* and the *how* of PKDD. These requirements follow from: the basic requirements imposed by KDD (Section 2), current KDD algorithmic techniques (Section 3), the trends in commodity hardware design (Section 4) and software requirements (Section 5). One difficulty in making such a survey is that each research community has its own jargon, which we will try to make accessible by describing it within a common PKDD framework.

## 2 PKDD Requirements

We begin by discussing the wish-list or desirable features of a functional PKDD system, using it to guide the rest of the survey. We mainly concentrate on aspects that have not received wide attention as yet.

*Algorithm Evaluation:* Algorithmic aspects that need attention are the ability to handle high dimensional datasets, to support terabyte data-stores, to minimize number of data scans, etc. An even more important research area is to provide a rapid development framework to implement and conduct the performance evaluation of a number of competing parallel methods for a given mining task. Currently this is a very time-consuming process, and there are no guidelines when to use a particular algorithm over another.

*Process Support:* The toolkit should support all KDD steps, from pre-processing operations for like sampling, discretization, and feature subset selection, to post-processing operations like rule grouping and pruning and model scoring. Other aspects include (persistent) pattern management operations like caching, efficient retrieval, and meta-level mining.

*Location Transparency:* The PKDD system should be able to seamlessly access and mine datasets regardless of their location, be they centralized or distributed.

*Data Type Transparency:* The system should be able to cope with heterogeneity (e.g., different database schemas), without having to materialize a join of multiple tables. Other difficult aspects deal with handling unstructured (hyper-)text, spreadsheet, and a variety of other data types.

*System Transparency:* This refers to the fact that the PKDD system should be able to seamlessly access file systems, databases, or data archives. Databases and data warehouses represent one kind of data repositories, and thus it is crucial to integrate mining with DBMS to avoid extracting data to flat files. On the other hand, a huge amount of data remains outside databases in flat-files, web-logs, etc. The PKDD system must therefore bridge the gap that exists today between the database and file-systems worlds (Choudhary and Kotz 1996). This is required since database systems today offer little functionality to support mining applications (Agrawal *et al.* 1993), and most research on parallel file systems and parallel I/O has looked at scientific applications, while data mining operations have very different workload characteristics.
*Security, QoS and Pricing:* In an increasingly networked world one constantly needs access to proprietary third-party and other remote datasets. The two main issues that need attention here are security and Quality-of-Service (QoS) issues in data mining. We need to prevent unauthorized mining, and we need to provide cost-sensitive mining to guarantee a level of performance. These issues are paramount in web-mining for e-commerce.
*Availability, Fault Tolerance and Mobility:* Distributed and parallel systems have more points of failure than centralized systems. Furthermore temporary disconnections (which are frequent in mobile computing environments) and reconnections by users should be tolerated with a minimal penalty to the user. Many real world applications cannot tolerate outages, and in the presence of QoS guarantees and contracts outages, can breach the agreements between providers and users. Little work has been done to address this area as well.

In the discussion below, due to space constraints, we choose to concentrate only on the algorithmic and hardware trends, and system transparency issues (i.e., parallel I/O and parallel and distributed databases), while briefly touching on other aspects (a more detailed paper is forthcoming).

## 3 Mining Methods

Faster and scalable algorithms for mining will always be required. Parallel and distributed computing seems ideally placed to address these big data performance issues. However, 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.

The parallel design space spans a number of systems and algorithmic components such as the hardware platform (shared vs. distributed), kind of parallelism (task vs. data), load balancing strategy (static vs. dynamic), data layout (horizontal vs. vertical) and search procedure used (complete vs. greedy).

Recent algorithmic work has been very successful in showing the benefits of parallelism for many of the common data mining tasks including association rules (Agrawal and Shafer 1996; Cheung *et al.* 1996; Han *et al.* 1997; Zaki *et al.* 1997), sequential patterns (Shintani and Kitsuregawa 1998; Zak-

i 2000), classification (Shafer *et al.* 1996; Joshi *et al.* 1998; Zaki *et al.* 1999; Sreenivas *et al.* 1999), regression (Williams *et al.* 2000) and clustering (Judd *et al.* 1996; Dhillon and Modha 2000; S. Goil and Choudhary 1999).

The typical trend in parallel mining is to start with a sequential method and pose various parallel formulations, implement them, and conduct a performance evaluation. While this is very important, it is a very costly process. After all, the parallel design space is vast and results on the parallelization of one serial method may not be applicable to other methods. The result is that there is a proliferation of parallel algorithms without any standardized benchmarking to compare and provide guidelines on which methods work better under what circumstances. The problem becomes even worse when a new and improved serial algorithm is found, and one is forced to come up with new parallel formulations. Thus, it is crucial that the PKDD system support rapid development and testing of algorithms to facilitate algorithmic performance evaluation.

One recent effort in this direction is discussed by (Skillicorn 1999). He emphasizes the importance of and presents a set of cost measures that can be applied to parallel algorithms to predict their computation, data access, and communication performance. These measures make it possible to compare different parallel implementation strategies for data-mining techniques without benchmarking each one.

A different approach is to build a data mining kernel that supports common data mining operations, and is modular in design so that new algorithms or their "primitive" components can be easily added to increase functionality. An example is the MKS (Anand *et al.* 1997) kernel. Also, generic set-oriented primitive operations were proposed in (Freitas and Lavington 1998) for classification and clustering, which were integrated with a parallel DBMS.

## 4 Hardware Models and Trends

The current hardware trends are that memory and disk capacity are increasing at a much higher rate than their speed. Furthermore, CPU capacity is roughly obeying Moore's law, which predicts doubling performance approximately every 18 months. To combat bus and memory bandwidth limitations, caching is used to improve the mean access time, giving rise to Non-Uniform Memory Access architectures. To accelerate the rate of computation, modern machines frequently increase the number of processing elements in an architecture. Logically, the memory of such machines is kept consistent, giving rise to a shared memory model, called Symmetric Multiprocessing (SMP) in the architecture community and *shared everything* in the database community (DeWitt and Gray 1992; Valduriez 1993). However the scalability of such architectures is limited, so for higher degrees of parallelism, a cluster of SMP nodes is used. This model, called *shared-nothing* in database literature, is also the preferred architecture for parallel databases (DeWitt and Gray 1992).

*Redundant arrays of independent (or inexpensive) disks* (RAID) (Chen *et al.* 1994) has gained popularity to increase I/O bandwidth and storage capacity,

reduce latency, and (optionally) support fault tolerance. In many systems, since the amount of data exceeds that which can be stored on disk, *tertiary storage* is used, typically consisting of one or more removable media devices with a juke box to swap the loaded media.

In addition to the current trends, there have been other ideas to improve the memory and storage bottlenecks. *Active Disks* (Riedel *et al.* 1997) and *Intelligent Disks* (Keeton *et al.* 1998) have been proposed as a means to exploit the improved processor performance of embedded processors in disk controllers to allow more complex I/O operations and optimizations, while reducing the amount of traffic over a congested I/O bus. Intelligent RAM (IRAM) (Kozyrakis and Patterson 1998) seeks to integrate processing elements in the memory. Active disks and IRAM are not currently prevalent, as the required hardware and systems software are not commonly available.

## 5 Software Infrastructure

Since our goal is to use commodity hardware, much of the support for our desired functionality is pushed back into the software. In this section we discuss some of the system transparency issues in PKDD systems, i.e., support for seamless access to databases and file systems and parallel I/O. We review selected aspects of these areas.

The most common database constructions currently in use are relational databases, object oriented databases, and object-relational databases. The data base layer ensures referential integrity and provides support for queries and/or transactions on the data set (Oszu and Valduriez 1999). The data base layer is frequently accessed via a query language, such as SQL. We are primarily interested in parallel and distributed database systems (DeWitt and Gray 1992; Valduriez 1993), which have data sets spanning disks. The primary advantages of such systems are that capacity of storage is improved and that parallelizing of disk access improves bandwidth and (for large I/O's) can reduce latency. Early on parallel database research explored special-purpose database machines for performance (Hsiao 1983), but today the consensus is that its better to use available parallel platforms, with shared-nothing paradigm as the architecture of choice. Shared-nothing database systems include Teradata, Gamma (D. DeWitt et al. 1990), Tandem (Tandem Performance Group 1988), Bubba (Boral *et al.* 1990), Arbre (Lorie *et al.* 1989), etc. We refer the reader to (DeWitt and Gray 1992; Valduriez 1993; Khan *et al.* 1999) for excellent survey articles on parallel and distributed databases. Issues within parallel database research of relevance to PKDD include the data partitioning (over disks) methods used, such as simple *round-robin* partitioning, where records are distributed evenly among the disks. *Hash partitioning* is most effective for applications requiring associative access since records are partitioned based on a hash function. Finally, *range partitioning* clusters records with similar attributes together. Most parallel data mining work to-date has used a round-robin approach to data partitioning. Other methods might be more suitable. Exploration of efficient multidimensional indexing

structures for PKDD is required (Gaede and Gunther 1998). The vast amount of work on parallel relational query operators, particularly parallel join algorithms, is also of relevance (Pirahesh *et al.* 1990). The use of DBMS *views* (Oszu and Valduriez 1999) to restrict the access of a DBMS user to a subset of the data, can be used to provide security in KDD systems.

Parallel I/O and file systems techniques are geared to handling large data sets in a distributed memory environment, and appear to be a better fit than distributed file systems for managing the large data sets found in KDD applications. Parallel File Systems and Parallel I/O techniques have been widely studied; (Kotz ) maintains an archive and bibliography, which has a nice reference guide (Stockinger 1998). Use of parallel I/O and file systems becomes necessary if RAID devices have insufficient capacity (due to scaling limitations) or contention for shared resources (e.g. buses or processors) exceeds the capacity of SMP architectures. The Scalable I/O initiative (SIO) includes many groups, including the *Message Passing Interface* (MPI) forum, which has adopted a MPI-IO API (Thakur *et al.* 1999) for parallel file management. MPI-IO is layered on top of local file systems. MPI uses a run time type definition scheme to define communication and I/O entity types. The ROMIO library (Thakur *et al.* 1999) implements MPI-IO in Argonne's MPICH implementation of MPI. ROMIO automates scheduling of aggregated I/O requests and uses the ADIO middleware layer to provide portability and isolate implementation dependent parts of MPIO. PABLO, another SIO member group, has created the *portable parallel file systems* (PPFS II), designed to support efficient access of large data sets in scientific applications with irregular access patterns. More information on parallel and distributed I/O and file systems appears in (Kotz ; Carretero *et al.* 1996; Gibson *et al.* 1999; Initiative ; Moyer and Sunderam 1994; Nieuwejaar and Kotz 1997; Schikuta *et al.* 1998; Seamons and Winslett 1996).

Users of PKDD systems are interested in maximizing performance. Prefetching is an important performance enhancing technique that can reduce the impact of latency by overlapping computation and I/O (Cortes 1999; Kimbrel *et al.* 1996; Patterson III 1997). In order for prefetching to be effective, the distributed system uses *hints* which indicate what data is likely to be used in the near future. Generation of accurate hints (not surprisingly) tends to be difficult since it relies on predicting a program's flow of control. Many hint generation techniques rely on traces of a program's I/O access patterns. (Kimbrel *et al.* 1996) surveyed a range of trace driven techniques and prefetching strategies, and provided performance comparisons. (Madhyastha and Reed 1997) recently used machine learning tools to analyze I/O traces from the PPFS, relying on artificial neural networks for on-line analysis of the current trace, and hidden markov models to analyze data obtained by profiling. (Chang and Gibson 1999) developed *SpecHint* which generates hints via speculative execution. We conjecture that PKDD techniques can be used to identify reference patterns, to provide hint generation and to address open performance analysis issues (Reed *et al.* 1998).

As we noted earlier, integration of various systems components for effective KDD is lagging. The current state of KDD tools can accurately be captured by

the term *flat-file mining*, i.e., prior to mining, all the data is extracted into a flat-file, which is then used for mining, effectively bypassing all database functionality. This is mainly because traditional databases are ill-equipped to handle/optimize the complex query structure of mining methods. However, recent work has recognized the need for integrating of the database, query management and data mining layers (Agrawal and Shim 1996; Sarawagi *et al.* 1998). (Agrawal and Shim 1996) postulated that better integration of the query manager, database and data mining layers would provide a speedup. (Sarawagi *et al.* 1998) confirmed that performance improvements could be attained, with the best performance obtained in *cache-mine* which caches and mines the query results on a local disk. SQL-like operators for mining association rules have also been developed (Meo *et al.* 1996). Further, proposals for data mining query language (Han *et al.* 1996; Imielinski and Mannila 1996; Imielinski *et al.* 1996; Siebes 1995) have emerged. We note that most of this work is targeted for serial environments. PKDD efforts will benefit from this research, but the optimization problems will of course be different in a parallel setting. Some exceptions include the parallel generic primitives proposed in (Freitas and Lavington 1998), and Data Surveyor (Holsheimer *et al.* 1996), a mining tool that uses the Monet database server for parallel classification rule induction. We further argue that we need a wider integration of parallel and distributed databases and file systems, to fully mine all available data (only a modest fraction of which actually resides in databases). Integration of PKDD and parallel file systems should enhance performance by improving hint generation in prefetching. Integrated PKDD can use parallel file systems for storing and managing large data sets and use distributed file system as an access point suited to mobile clients for management of query results.

## 6 Conclusions

We described a list of desirable design features of parallel KDD systems. These requirements motivated a brief survey of existing algorithmic and systems support for building such large-scale mining tools. We focused on the state-of-the-art in databases, and parallel I/O techniques. We observe that implementing a effective PKDD system requires integration of these diverse sub-fields into a coherent and seamless system. Emerging issues in PKDD include benchmarking, security, availability, mobility and QoS, motivating fresh research in these disciplines. Finally, PKDD approaches may be used as a tool in these areas (e.g. hint generation for prefetching in parallel I/O), resulting in a bootstrapping approach to software development.

# References


R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Trans. on Knowledge and Data Engg.*, 8(6):962–969, December 1996.

R. Agrawal and K. Shim. Developing tightly-coupled data mining applications on a relational DBMS. In *Int'l Conf. on Knowledge Discovery and Data Mining*, 1996.

R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Trans. on Knowledge and Data Engg.*, 5(6):914–925, December 1993.

S. Anand, *et al.* Designing a kernel for data mining. *IEEE Expert*, pages 65–74, March 1997.

H. Boral, *et al.* Prototyping Bubba, a highly parallel database system. *IEEE Trans. on Knowledge and Data Engg.*, 2(1), March 1990.

J. Carretero, *et al.* ParFiSys: A parallel file system for MPP. *ACM Operating Systems Review*, 30(2):74–80, 1996.

F. Chang and G. Gibson. Automatic i/o hint generation through speculative execution. In *Symp. on Operating Systems Design and Implementation*, February 1999.

P. M. Chen, *et al.* RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

D. Cheung, *et al.* A fast distributed algorithm for mining association rules. In *4th Int'l Conf. Parallel and Distributed Info. Systems*, December 1996.

A. Choudhary and D. Kotz. Large-scale file systems with the flexibility of databases. *ACM Computing Surveys*, 28A(4), December 1996.

T. Cortes. *High Performance Cluster Computing*, Vol. 1, chapter Software Raid and Parallel File Systems, pages 463–495. Prentice Hall, 1999.

D. DeWitt et al. The GAMMA database machine project. *IEEE Trans. on Knowledge and Data Engg.*, 2(1):44–62, March 1990.

D. DeWitt and J. Gray. Parallel database systems: The future of high-performance database systems. *Communications of the ACM*, 35(6):85–98, June 1992.

I. S. Dhillon and D. S. Modha. A clustering algorithm on distributed memory machines. In *Zaki and Ho, 2000.*

A. Freitas and S. Lavington. *Mining very large databases with parallel processing*. Kluwer Academic Pub., 1998.

V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

G. Gibson, *et al.* NASD scalable storage systems. In *USENIX99, Extreme Linux Workshop*, June 1999.

J. Han, *et al.* DMQL: A data mining query language for relational databases. In *SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, June 1996.

E-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *ACM SIGMOD Conf. Management of Data*, May 1997.

M. Holsheimer, M. L. Kersten, and A. Siebes. Data surveyor: Searching the nuggets in parallel. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1996.

D. Hsiao. *Advanced Database Machine Architectures*. Prentice Hall, 1983.

T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11), November 1996.

T. Imielinski, A. Virmani, and A. Abdulghani. DataMine: Application programming interface and query language for database mining. In *Int'l Conf. Knowledge Discovery and Data Mining*, August 1996.

Scalable I/O Initiative. *http://www.cacr.caltech.edu/SIO*. California Institute of Technology.

M. Joshi, G. Karypis, and V. Kumar. ScalParC: A scalable and parallel classification algorithm for mining large datasets. In *Int'l Parallel Processing Symposium*, 1998.

D. Judd, P. McKinley, and A. Jain. Large-scale parallel data clustering. In *Int'l Conf. Pattern Recognition*, 1996.

H. Kargupta and P. Chan, editors. *Advances in Distributed Data Mining*. AAAI Press, 2000.

K. Keeton, D. Patterson, and J.M. Hellerstein. The case for intelligent disks. *SIGMOD Record*, 27(3):42–52, September 1998.

M.F. Khan, *et al.* Intensive data management in parallel systems: A survey. *Distributed and Parallel Databases*, 7:383–414, 1999.

T. Kimbrel, *et al.* A trace-driven comparison of algorithms for parallel prefetching and caching. In *USENIX Symp. on Operating Systems Design and Implementation*, pages 19–34, October 1996.

D. Kotz. The parallel i/o archive. Includes pointers to his Parallel I/O Bibliography, can be found at *http://www.cs.dartmouth.edu/pario/*.

C. E. Kozyrakis and D. A. Patterson. New direction in computer architecture research. *IEEE Computer*, pages 24–32, November 1998.

R. Lorie, *et al.* Adding inter-transaction parallelism to existing DBMS: Early experience. *IEEE Data Engineering Newsletter*, 12(1), March 1989.

T. M. Madhyastha and D. A. Reed. Exploiting global input/output access pattern classification. In *Proceedings of SC'97*, 1997. On CDROM.

R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. In *Int'l Conf. Very Large Databases*, 1996.

S. A. Moyer and V. S. Sunderam. PIOUS: a scalable parallel I/O system for distributed computing environments. In *Scalable High-Performance Computing Conf.*, 1994.

N. Nieuwejaar and D. Kotz. The galley parallel file system. *Parallel Computing*, 23(4), June 1997.

M. T. Oszu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

R. H. Patterson III. *Informed Prefetching and Caching*. PhD thesis, Carnegie Mellon University, December 1997.

Pirahesh *et al. Parallelism in Relational Data Base Systems.* In *nt'l Symp. on Parallel and Distributed Systems*, July 1990.

D. A. Reed, *et al.* Performance analysis of parallel systems: Approaches and open problems. In *Joint Symposium on Parallel Processing (JSPP)*, June 1998.

E. Riedel, G. A. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia. In *Int'l Conf. on Very Large Databases*, August 1997.

H. Nagesh S. Goil and A. Choudhary. MAFIA: Efficient and scalable subspace clustering for very large data sets. Technical Report 9906-010, Northwestern University, June 1999.

S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with databases: alternatives and implications. In *ACM SIGMOD Conf. on Management of Data*, June 1998.

E. Schikuta, T. Fuerle, and H. Wanek. ViPIOS: The vienna parallel input/output system. In *Euro-Par'98*, September 1998.

K. E. Seamons and M. Winslett. Multidimensional array I/O in Panda 1.0. *Journal of Supercomputing*, 10(2):191–211, 1996.

J. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *Int'l Conf. on Very Large Databases*, March 1996.

T. Shintani and M. Kitsuregawa. Mining algorithms for sequential patterns in parallel: Hash based approach. In *2nd Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, April 1998.

A. Siebes. Foundations of an inductive query language. In *Int'l Conf. on Knowledge Discovery and Data Mining*, August 1995.

D. Skillicorn. Strategies for parallel data mining. *IEEE Concurrency*, 7(4):26–35, October-December 1999.

M. Sreenivas, K. Alsabti, and S. Ranka. Parallel out-of-core divide and conquer techniques with application to classification trees. In *Int'l Parallel Processing Symposium*, April 1999.

H. Stockinger. Dictionary on parallel input/output. Master's thesis, Dept. of Data Engineering, University of Vienna, February 1998.

Tandem Performance Group. A benchmark of non-stop SQL on the debit credit transaction. In *SIGMOD Conference*, June 1988.

R. Thakur, W. Gropp, and E. Lusk. On implementing mpi-io portably and with high performance. In *Workshop on I/O in Parallel and Distributed Systems*, May 1999.

P. Valduriez. Parallel database systems: Open problems and new issues. *Distributed and Parallel Databases*, 1:137–165, 1993.

G. Williams, *et al.* The integrated delivery of large-scale data mining: The ACSys data mining project. In *Zaki and Ho. 2000.*

M. J. Zaki and C-T. Ho, editors. *Large-Scale Parallel Data Mining*, LNCS Vol. 1759. Springer-Verlag, 2000.

M. J. Zaki, *et al.* Parallel algorithms for fast discovery of association rules. *Data Mining and Knowledge Discovery: An International Journal*, 1(4):343-373, December 1997.

M. J. Zaki, C.-T. Ho, and R. Agrawal. Parallel classification for data mining on shared-memory multiprocessors. In *Int'l Conf. on Data Engineering*, March 1999.

M. J. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency*, 7(4):14–25, 1999.

M. J. Zaki. Parallel sequence mining on SMP machines. In *Zaki and Ho, 2000.*

# Parallel Data Mining on ATM-Connected PC Cluster and Optimization of its Execution Environments

Masato OGUCHI[1,2] and Masaru KITSUREGAWA[1]

[1] Institute of Industrial Science, The University of Tokyo
7-22-1 Roppongi, Minato-ku Tokyo 106-8558, Japan
[2] Informatik4, Aachen University of Technology
Ahornstr.55, D-52056 Aachen, Germany
oguchi@tkl.iis.u-tokyo.ac.jp

**Abstract.** In this paper, we have constructed a large scale ATM-connected PC cluster consists of 100 PCs, implemented a data mining application, and optimized its execution environment. Default parameters of TCP retransmission mechanism cannot provide good performance for data mining application, since a lot of collisions occur in the case of all-to-all multicasting in the large scale PC cluster. Using a TCP retransmission parameters according to the proposed parameter optimization, reasonably good performance improvement is achieved for parallel data mining on 100 PCs.
Association rule mining, one of the best-known problems in data mining, differs from conventional scientific calculations in its usage of main memory. We have investigated the feasibility of using available memory on remote nodes as a swap area when working nodes need to swap out their real memory contents. According to the experimental results on our PC cluster, the proposed method is expected to be considerably better than using hard disks as a swapping device.

## 1 Introduction

Looking over the recent technology trends, PC/WS clusters connected with high speed networks such as ATM are considered to be a principal platform for future high performance parallel computers. Applications which formerly could only be implemented on expensive massively parallel processors can now be executed on inexpensive clusters of PCs. Various research projects to develop and examine PC/WS clusters have been performed until now[1][2][3]. Most of them however, only measured basic characteristics of PCs and networks, and/or some small benchmark programs were examined. We believe that data intensive applications such as data mining and ad-hoc query processing in databases are quite important for future high performance computers, in addition to the conventional scientific applications[4].

Data mining has attracted a lot of attention recently from both the research and commercial community, for finding interesting trends hidden in large transaction logs. Since data mining is a very computation and I/O intensive process,

parallel processing is required to supply the necessary computational power for very large mining operations. In this paper, we report the results on parallel data mining on ATM-connected PC clusters, consists of 100 Pentium Pro PCs.

## 2 Our ATM-connected PC cluster and its communication characteristics

We have constructed a PC cluster pilot system which consists of 100 nodes of 200MHz Pentium Pro PCs connected with an ATM switch. An overview of the PC cluster is shown in Figure 1. Each node of the cluster is equipped with 64Mbytes main memory, 2.5Gbytes IDE hard disk, and 4.3Gbytes SCSI hard disk. Solaris ver.2.5.1 is used as an operating system.

All nodes of the cluster are connected by a 155Mbps ATM LAN as well as an Ethernet. HITACHI's AN1000-20, which has 128 port 155Mbps UTP-5, is used as an ATM switch. Interphase 5515 PCI ATM adapter and RFC-1483 PVC driver, which support LLC/SNAP encapsulation for IP over ATM, are used. Only UBR traffic class is supported in this driver.

TCP/IP is used as a communication protocol. TCP is not only a very popular reliable protocol for computer communication, but also contains all functions as a general transport layer. Thus the results of our experiments must be valid even if other transport protocol is used, for investigating reliable communication protocols on a large scale cluster.

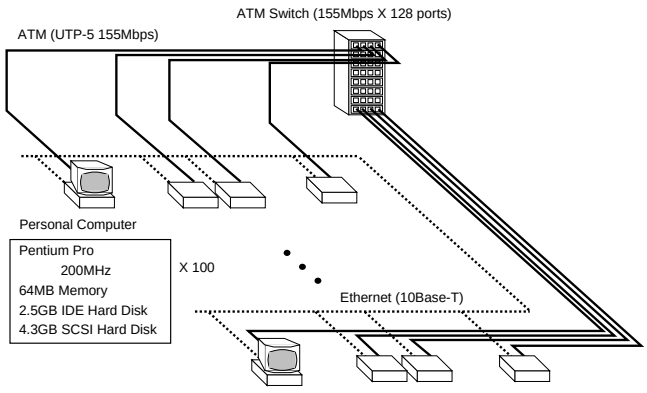



**Fig. 1.** An overview of the PC cluster

## 3 Parallel data mining application and its implementation on the cluster

### 3.1 Association rule mining

Data mining is a method of the efficient discovery of useful information such as rules and previously unknown patterns existing among data items embedded in large databases, which allows more effective utilization of existing data. One of the best known problems in data mining is mining of the association rules from a database, so called "basket analysis"[5][6]. Basket type transactions typically consist of transaction id and items bought per-transaction. An example of an association rule is "if customers buy A and B then 90% of them also buy C". The best known algorithm for association rule mining is Apriori algorithm proposed by R. Agrawal of IBM Almaden Research[7].

In order to improve the quality of the rule, we have to analyze very large amounts of transaction data, which requires considerably long computation time. We have studied several parallel algorithms for mining association rules until now[8], based on Apriori. One of these algorithms, called HPA(Hash Partitioned Apriori), is implemented and evaluated.

Apriori first generates candidate itemsets, then scans the transaction database to determine whether the candidates satisfy the user specified minimum support. At first pass (pass 1), a support for each item is counted by scanning the transaction database, and all items which satisfy the minimum support are picked out. These items are called large 1-itemsets. In the second pass (pass 2), 2-itemsets (length 2) are generated using the large 1-itemsets. These 2-itemsets are called candidate 2-itemsets. Then supports for the candidate 2-itemsets are counted by scanning the transaction database, large 2-itemsets which satisfy the minimum support are determined. This iterative procedure terminates when large itemset or candidate itemset becomes empty. Association rules which satisfy user specified minimum confidence can be derived from these large itemsets.

HPA partitions the candidate itemsets among processors using a hash function, like the hash join in relational databases. HPA effectively utilizes the whole memory space of all the processors. Hence it works well for large scale data mining. In the detail of HPA, please refer to [8][9].

### 3.2 Implementation of HPA program on PC cluster

We have implemented HPA program on our PC cluster. Each node of the cluster has a transaction data file on its own hard disk. Solaris socket library is used for the inter-process communication. All processes are connected with each other by socket connections, thus forming mesh topology. In the ATM level, PVC (Permanent Virtual Channel) switching is used since the data is transferred continuously among all the processes.

Transaction data is produced using data generation program developed by Agrawal, designating some parameters such as the number of transaction, the

**Table 1.** The number of candidate and large itemsets

| | |
|---|---|
| $C$ | the number of candidate itemsets |
| $L$ | the number of large itemsets |
| $T$ | the execution time of each pass [sec] |

| pass | $C$ | $L$ | $T$ |
|---|---|---|---|
| pass 1 | | 1023 | 11.2 |
| pass 2 | 522753 | 32 | 69.8 |
| pass 3 | 19 | 19 | 3.2 |
| pass 4 | 7 | 7 | 6.2 |
| pass 5 | 1 | 0 | 12.1 |

number of different items, and so on. The produced data is divided by the number of nodes, and copied to each node's hard disk.

The parameters used in the evaluation is as follows: The number of transaction is 10,000,000, the number of different items is 5000 and minimum support is 0.7%. The size of the transaction data is about 800Mbytes in total. The message block size is set to be 8Kbytes and the disk I/O block size is 64Kbytes.

The numbers of candidate itemsets and large itemsets, and the execution time of each pass executed on 100 nodes PC cluster are shown in Table 1. Note that the number of candidate itemsets in pass 2 is extremely larger than other passes, which often happens in association rules mining.

## 4 Optimization of transport layer protocol parameters

### 4.1 Broadcasting on the cluster and TCP retransmission

The execution times of pass $3 - 5$ are relatively long in Table 1, although they do not have large number of itemsets. At the end of each pass, a barrier synchronization and exchange of data are needed among all nodes, that is, all-to-all broadcasting takes place. Even if the amount of broadcasting data is not large, cells must be discarded at the ATM switch if timing of the broadcasting is the same at all nodes. Since pass $3 - 5$ have little data to process, actual execution time is quite short, thus broadcasting is performed almost simultaneously in all nodes, which tend to cause network congestion and TCP retransmission as a result. We have executed several experiments to find the better retransmission parameters setting suitable for such cases.

We use TCP protocol implemented in Solaris OS, whose parameters can be changed with user level commands. Two parameters changed here are 'maximum interval of TCP retransmission' and 'minimum interval of TCP retransmission', which we call 'MAX' and 'MIN' respectively. The default setting is MAX = 60000 [msec] and MIN = 200 [msec] in the current version of Solaris. The interval of

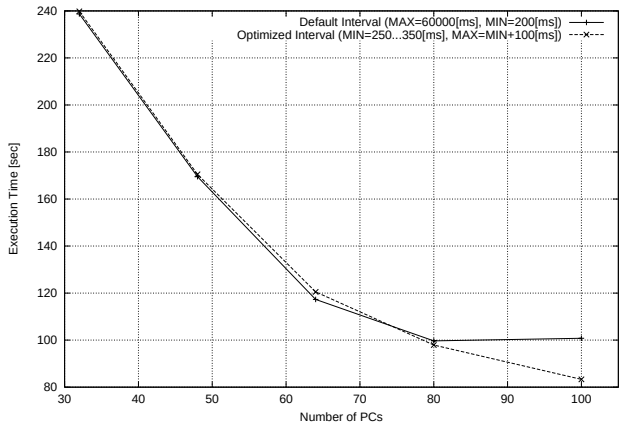



**Fig. 2.** Execution time of HPA program on PC cluster

TCP retransmission is dynamically changed in the protocol, within the limits between MAX and MIN.

As a result of experiments, we have found that the default value of MAX is not suitable for the cluster, which might cause the unnecessary long retransmission interval. MAX should be set to be smaller than the default value, such as MAX = MIN + 100[msec]. Moreover, MIN is better to be set as random value, which can prevent the collision of the cells at ATM switch.

### 4.2 Total performance of HPA program using proposed method

HPA program is executed using the proposed parameter setting of TCP on the PC cluster pilot system. The execution time of HPA program is shown in Figure 2. In this Figure, one line indicates the case using default TCP retransmission parameters, i.e. MAX = 60000[msec] and MIN = 200[msec], and the other line indicates the case using random parameters (MIN = 250 ... 350[ms], MAX = MIN + 100[ms]).

Reasonably good speedup is achieved up to 100 PCs using proposed optimized parameters. Since the application itself is not changed, the difference only comes from TCP retransmission, occurred along with barrier synchronization and all-to-all data broadcasting.

## 5 Dynamic remote memory acquisition

### 5.1 Dynamic remote memory acquisition and its experiments

As shown in section 3, the number of candidate itemsets in pass 2 is very much larger than in other passes in association rule mining. The number of itemsets is

strongly dependent on user-specified conditions, such as the minimum support value, and it is difficult to predict how large the number will be before execution. Therefore, it may happen that the number of candidate itemsets increases dramatically in this step so that the memory requirement becomes extremely large. When the required memory is larger than the real memory size, part of the contents of memory must be swapped out. However, because the size of each data item is rather small and all the data is accessed almost at random, swapping out to a storage device is expected to degrade the total performance severely.

We have executed several experiments in which available memory in remote nodes is used as a swap area when huge memory is dynamically required. In the experiments, a limit value for memory usage of candidate itemsets is set at each node. When the amount of memory used exceeds this value during the execution of the HPA program(in Pass 2), part of the contents is swapped out to available memory in remote nodes, that is, application execution nodes acquire remote memory dynamically. Although such available remote nodes could be found dynamically in a real system, we selected them statically in these experiments. On the other hand, when an application execution node trys to access an item that had been swapped out, a pagefault occurs.

The basic behavior of this approach has something in common with distributed shared memory systems[10], memory management system in distributed operating systems[11], or cache mechanism in client-server database systems[12]. For example, if data structures inside applications are considered in distributed shared memory, almost the same effect can be expected. That is to say, it is possible to program almost the same mechanism using some types of distributed shared memory systems. Thus, our mechanism might be regarded as equivalent to a case of distributed shared memory optimized for a particular application.

We have executed experiments of the proposed mechanism on the PC cluster. The parameters used in the experiment are as follows. The number of transactions is 1,000,000, the number of different items is 5,000, and the minimum support is 0.1%. The number of application execution nodes is 8 in this evaluation. The number of memory available nodes is varied from 1 to 16. With these conditions, the total number of candidate itemsets in pass 2 is 4,871,881. Since each candidate itemset occupied 24 bytes in total(structure area + data area), approximately 14-15Mbytes of memory were filled with these candidate itemsets at each node.

### 5.2 Remote update method

When memory usage is limited, the execution time is much longer than when there is no memory limit. This is because the number of swapouts is extremely large. In Table 2 the numbers of pagefaults on each application execution node are shown. Because most of the memory contents are accessed repeatedly, a kind of thrashing seems to happen in these cases. In order to prevent this phenomenon, a method for restricting swapping operations is proposed.

When usage of memory reaches the limit value at a node, it acquires remote memory and swaps out part of its memory contents. The contents will be

**Table 2.** The numbers of pagefaults on each application node

| Usage limit | node 1 | node 2 | node 3 | node 4 | node 5 | node 6 | node 7 | node 8 |
|---|---|---|---|---|---|---|---|---|
| 12[MB] | 1606258 | 2925254 | 1306521 | 2361756 | 1671840 | 1723410 | 2166277 | 2545003 |
| 13[MB] | 885798 | 1896226 | 593000 | 1374688 | 932374 | 896150 | 1326941 | 1375398 |
| 14[MB] | 254094 | 1003757 | | 512984 | 286945 | 191102 | 601657 | 407628 |
| 15[MB] | | 268039 | | | | | | |

swapped in again if this data is accessed later. Instead of swapping, it is sometimes better to send update information to the remote memory when a pagefault occurs. That is to say, once some contents are swapped out to memory in a distant node, they are fixed there and accessed only through a remote memory access interface provided by library functions. This remote update method has been applied only to the itemsets counting phase, for simplicity.

The access interface function has been developed to realize the remote update operations. The execution time using this method is shown in Figure 3. This figure shows the execution time of pass 2 of the HPA program, when the number of memory available nodes is 16. The execution times for dynamic remote memory acquisition, according to this method and the previous simple swapping case, are compared in the Figure. The execution time using hard disks as a swapping device is also shown, for comparison. Seagate Barracuda 7,200[rpm] SCSI hard disks have been used for this purpose. Other conditions are the same as the case of dynamic remote memory acquisition.

The execution time using hard disks as swapping devices is very long, especially when the memory usage limit is small, because each access time to a hard disk is much longer than that for remote memory through the network. The execution time of dynamic remote memory acquisition with simple swapping is better than for swapping out to hard disks. It increases, however, when the memory usage limit is small, since the number of pagefaults becomes extremely large in such a case.

Compared to these results, the execution time of dynamic remote memory acquisition with remote update operations is quite short, even when the memory usage limit is small. It seems to be effective to provide a simple remote access interface for the itemsets counting phase, because the number of swapping operations during this phase is very large. These results indicate that, performance of the proposed remote memory acquisition with remote update operations is considerably better than other methods.

## References

1. C. Huang and P. K. McKinley: "Communication Issues in Parallel Computing Across ATM Networks", *IEEE Parallel and Distributed Technology*, Vol.2, No.4, pp.73-86, 1994.

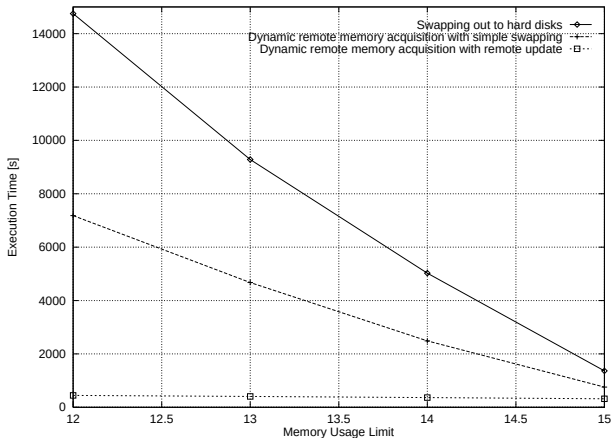



**Fig. 3.** Comparison of proposed methods


2. R. Carter and J. Laroco: "Commodity Clusters: Performance Comparison Between PC's and Workstations", *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, pp.292-304, August 1996.
3. D. E. Culler et al.: "Parallel Computing on the Berkeley NOW", *Proceedings of the 1997 Joint Symposium on Parallel Processing(JSPP '97)*, pp.237-247, May 1997.
4. T. Tamura, M. Oguchi, and M. Kitsuregawa: "Parallel Database Processing on a 100 Node PC Cluster: Cases for Decision Support Query Processing and Data Mining", *Proceedings of SuperComputing '97*, November 1997.
5. U. M. Fayyad et al.: "Advances in Knowledge Discovery and Data Mining", *The MIT Press*, 1996.
6. V. Ganti, J. Gehrke, and R. Ramakrishnan: "Mining Very Large Databases", *IEEE Computer*, Vol.32, No.8, pp.38-45, August 1999.
7. R. Agrawal, T. Imielinski, and A. Swami: "Mining Association Rules between Sets of Items in Large Databases", *Proceedings of the ACM International Conference on Management of Data*, pp.207-216, May 1993.
8. T. Shintani and M. Kitsuregawa: "Hash Based Parallel Algorithms for Mining Association Rules", *Proceedings of the Fourth IEEE International Conference on Parallel and Distributed Information Systems*, pp.19-30, December 1996.
9. M. J. Zaki: "Parallel and Distributed Association Mining: A Survey", *IEEE Concurrency*, Vol.7, No.4, pp.14-25, 1999.
10. C. Amza et al.: "TreadMarks: Shared Memory Computing on Networks of Workstations", *IEEE Computer*, Vol.29, No.2, pp.18-28, February 1996.
11. M. J. Feeley et al.: "Implementing Global Memory Management in a Workstation Cluster", *Proceedings of the ACM Symposium on Operating Systems Principles*, pp.201-212, December 1995.
12. S. Dar et al.: "Semantic Data Caching and Replacement", *Proceedings of 22nd VLDB Conference*, September 1996.

# The Parallelization of a Knowledge Discovery System with Hypergraph Representation*

Jennifer Seitzer, James P. Buckley, Yi Pan, and Lee A. Adams

Department of Computer Science
University of Dayton, Dayton, OH 45469-2160

**Abstract.** Knowledge discovery is a time-consuming and space intensive endeavor. By distributing such an endeavor, we can diminish both time and space. System **INDED**(pronounced "indeed") is an inductive implementation that performs rule discovery using the techniques of inductive logic programming and accumulates and handles knowledge using a deductive nonmonotonic reasoning engine. We present four schemes of transforming this large serial inductive logic programming (ILP) knowledge-based discovery system into a distributed ILP discovery system running on a Beowulf cluster. We also present our data partitioning algorithm based on locality used to accomplish the data decomposition used in the scenarios.

## 1 Introduction

Knowledge discovery in databases has been defined as the non-trivial process of identifying valid, novel, potentially useful, and understandable patterns in data [PSF91]. Data mining is a commonly used knowledge discovery technique that attempts to reveal patterns within a database in order to exploit implicit information that was previously unknown [CHY96]. One of the more useful applications of data mining is to generate all significant associations between items in a data set [AIS93]. A discovered pattern is often denoted in the form of an IF-THEN rule (IF *antecedent* THEN *consequent*), where the antecedent and consequent are logical conjunctions of predicates (first order logic) or propositions (propositional logic) [Qui86]. Graphs and hypergraphs are used extensively as knowledge representation constructs because of their ability to depict causal chains or networks of implications by interconnecting the consequent of one rule to the antecedent of another.

In this work, using the language of logic programming, we use a hypergraph to represent the knowledge base from which rules are mined. Because the hypergraph gets inordinantly large in the serial version of our system [Sei99], we have devised a parallel implementation where, on each node, a smaller sub-hypergraph is created. Consequently, because there is a memory limit to the size of a storable hypergraph, by using this parallel version, we are able to grapple with problems

* This work is partially supported under Grant 9806184 of the National Science Foundation.

involving larger knowledge bases than those workable on the serial system. A great deal of work has been done in parallelizing unguided discovery of association rules originally in [ZPO97] and recently refined in [SSC99]. The novel aspects of this work include the parallelization of both a nonmonotonic reasoning system and an ILP learner. In this paper, we present the schemes we have explored and are currently exploring in this pursuit.

## 2 Serial System INDED

System **INDED** is a knowledge discovery system that uses inductive logic programming (ILP) [LD94] as its discovery technique. To maintain a database of background knowledge, **INDED** houses a deduction engine that uses deductive logic programming to compute the current state (current set of true facts) as new rules and facts are procured.

### 2.1 Inductive Logic Programming

Inductive logic programming (ILP) is a new research area in artificial intelligence which attempts to attain some of the goals of machine learning while using the techniques, language, and methodologies of logic programming. Some of the areas to which ILP has been applied are data mining, knowledge acquisition, and scientific discovery [LD94]. The goal of an inductive logic programming system is to output a rule which *covers* (entails) an entire set of positive observations, or examples, and *excludes* or *does not cover* a set of negative examples [Mug92]. This rule is constructed using a set of known facts and rules, knowledge, called domain or *background* knowledge. In essence, the ILP objective is to synthesize a logic program, or at least part of a logic program using examples, background knowledge, and an entailment relation. The following definitions are from [LD94].

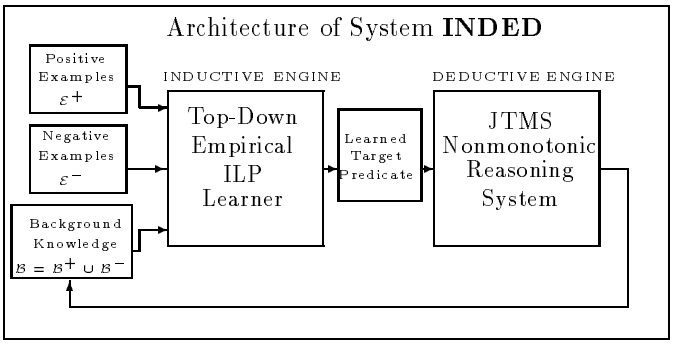
**Definition 2.1 (coverage, completeness, consistency)** *Given background knowledge $\mathcal{B}$, hypothesis $\mathcal{H}$, and example set $\mathcal{E}$, hypothesis $\mathcal{H}$* **covers** *example $e \in \mathcal{E}$ with respect to $\mathcal{B}$ if $\mathcal{B} \cup \mathcal{H} \models e$. A hypothesis $\mathcal{H}$ is* **complete** *with respect to background $\mathcal{B}$ and examples $\mathcal{E}$ if all positive examples are covered, i.e., if for all $e \in \mathcal{E}^+$ , $\mathcal{B} \cup \mathcal{H} \models e$. A hypothesis $\mathcal{H}$ is* **consistent** *with respect to background $\mathcal{B}$ and examples $\mathcal{E}$ if no negative examples are covered, i.e., if for all $e \in \mathcal{E}^-$ , $\mathcal{B} \cup \mathcal{H} \not\models e$.*

**Definition 2.2 (Formal Problem Statement)** *Let $\mathcal{E}$ be a set of training examples consisting of* **true** *$\mathcal{E}^+$ and* **false** *$\mathcal{E}^-$ ground facts of an unknown (target) predicate $\mathcal{T}$. Let $\mathcal{L}$ be a description language specifying syntactic restrictions on the definition of predicate $\mathcal{T}$. Let $\mathcal{B}$ be background knowledge defining predicates $q_i$ which may be used in the definition of $\mathcal{T}$ and which provide additional information about the arguments of the examples of predicate $\mathcal{T}$. The ILP problem is to produce a definition $\mathcal{H}$ for $\mathcal{T}$, expressed in $\mathcal{L}$, such that $\mathcal{H}$ is complete and consistent with respect to the examples $\mathcal{E}$ and background knowledge $\mathcal{B}$. [LD94]*

### 2.2 Serial Arichitecture

System **INDED** (pronounced "indeed") is comprised of two main computation engines. The deduction engine is a bottom-up reasoning system that computes the current state by generating a stable model [2], if there is one, of the current ground instantiation represented internally as a hypergraph, and by generating the well-founded model [VRS91], if there is no stable model[GL90]. This deduction engine is, in essence, a justification truth maintenance system which accommodates non-monotonic updates in the forms of positive or negative facts.

The induction engine, using the current state created by the deduction engine as the background knowledge base, along with positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$ , induces a rule(s) which is then used to augment the deductive engine's hypergraph. We use a standard top-down hypothesis construction algorithm (learning algorithm) in **INDED**[LD94]. This algorithm uses two nested programming loops. The outer (covering) loop attempts to cover all positive examples, while the inner loop (specialization) attempts to exclude all negative examples. Termination is dictated by two user-input values to indicate sufficiency and necessity stopping criterea. The following diagram illustrates the discovery constituents of **INDED** and their symbiotic interaction.




The input files to **INDED** that initialize the system are the extensional database (EDB) and the intensional database (IDB). The EDB is made up of initial ground facts (facts with no variables, only constants). The IDB is made up of universally quantified rules with no constants, only variables. Together, these form the internal ground instantiation represented internally as the deduction engine hypergraph.

## 3 Parallelizing INDED

Our main goals in parallelizing **INDED** are to obtain reasonably accurate rules faster and to decrease the size of the internal deduction hypergraph so that the

[2] Although the formal definitions of these semantics are cited above, for this paper, we can intuitively accept stable and well-founded models as those sets of facts that are generated by transitively applying modus ponens to rules.

problem space is increased. The serial version is very limited in what problems it can solve because of memory limitations. Work has been done in the direct partitioning of a hypergraph [KAK99]. In our pursuit to parallelize **INDED**, however, we are exploring the following schemes where indirect hypergraph reductions are performed. Each of the following scenarios has been devised to be implemented on a Beowulf cluster [Buy99] using MPI [GLS99]:

1. large grained control parallel decomposition where one node runs the induction engine while another node runs the deduction engine
2. large grained control parallel decomposition where a pipeline of processors are established each operating on a different current state as created in previous (or subsequent) pipelined iterations
3. data parallel decomposition where each node runs the same program with smaller input files (hence smaller internal hypergraphs).
4. speculative parallel approach where each node attempts to learn the same rule using a different predicate ranking alogrithm in the induction engine.

## 4 Naive Decomposition

In this decomposition, we create a very coarse grain system in which two nodes share the execution. One node houses the deduction engine; the other houses the induction. Our strategy lets the induction engine initially discover a target predicate from positive and negative examples and an initial background knowledge base. Meanwhile, the deduction engine computes the current state using initial input files. This current state is sent to the induction engine as its background knowledge base in the subsequent iteration. The learned predicate from the induction engine from one iteration is then fed into the deductive engine to be used during the next iteration in its computation of the current state. This is then used as the background knowledge for the induction engine during the subsequent iteration.

In general, during iteration $i$, the induction engine computes new intensional rules for the deduction engine to use in its computation of the current state in iteration $i + 1$. Simultaneously, during iteration $i$, the deduction engine computes a new current state for the induction engine to use as its background knowledge in iteration $i + 1$. The above process is repeated until all target predicates specified have been discovered. As we extend this implementation, we expect to acquire a pipe-lined system where the deduction engine is computing state $S_{i+1}$ while the induction engine is using $S_i$ to induce new rules (where $i$ is the current iteration number).

## 5 Data Parallel Decomposition with Data Partitioning

In this method, each worker node runs **INDED** when invoked by a master MPI node [GLS99]; each worker executes by running a partial background knowledge

base which, as in the serial version, is spawned by its deduction engine. In particular, each worker receives the full serial intensional knowledge base (IDB) but only a partial extensional knowledge base (EDB). The use of a partial EDB creates a significantly smaller (and different) hypergraph on each Beowulf worker node. This decomposition led to a faster execution due to a significantly smaller internal hypergraph being built. The challenge was to determine the best way to decompose the serial large EDB into smaller EDB's so that the rules obtained were as accurate as those learned by the serial version.

### 5.1 Data Partitioning and Locality

In this data parallel method, our attention centered on decomposition of the input files to reduce the size of any node's deduction hypergraph. We found that in many cases data transactions exhibited a form of locality of reference. Locality of reference is a phenomenon ardently exploited by cache systems where the general area of memory referenced by sequential instructions tends to be repeatedly accessed. Locality of reference in the context of knowledge discovery also exists and should be exploited to increase the efficiency of rule mining. A precept of knowledge discovery is that data in a knowledge base system are nonrandom and tend to cluster in a somewhat predictable manner. This tendency mimics locality of reference. There are three types of locality of reference which may coexist in a knowledge base system: spatial, temporal, and functional. In spatial locality of reference, certain data items appear together in a physical section of a database. In temporal locality of reference, the data items that are used in the recent past appear in the near future. For example, if there is a sale in a supermarket for a particular brand of toothpaste on Monday, we will see a lot of sales for this brand of toothpaste on that day. In functional locality of reference, we appeal to a semantic relationship between entities that have a strong semantic tie that affects data transactions relating to them. For example, cereal and milk are two semantically related objects. Although they are typically located in different areas of a store, many purchase transactions of one, include the other. All three of these localities can be exploited in distributed knowledge mining, and help justify the schemes adpoted in our implementations discussed in the following sections.

### 5.2 Partitioning Algorithm

To retain all global dependencies among the predicates in the current state, all Beowulf nodes receive a full copy of the serial IDB. The serial EDB, the initial large set of facts, therefore, is decomposed and partitioned among the nodes. The following algorithm transforms a large serial extensional database (EDB) into $p$ smaller EDB's to be placed on $p$ Beowulf nodes. It systematically creates sets based on constants appearing in the positive example set $\mathcal{E}^+$ . Some facts from the serial EDB could appear on more than one processor. The algorithm is of linear complexity requiring only one scan through the serial EDB and positive example set E+.

**Algorithm 5.1 (EDB Partitioning Algorithm)** This algorithm is $O(n)$, where $n$ is the number of facts in the EDB.

**Input:** Number of processors $p$ in Beowulf
Serial extensional database (EDB)
Positive and negative example set $\mathcal{E}^+$ , $\mathcal{E}^-$
**Output:** $p$ individual worker node EDB's

*BEGIN ALGORITHM 5.1*
For each example $e \in \mathcal{E}^+ \cup \mathcal{E}^-$ Do
For each constant $c \in e$ Do
create an initially empty set $S_c$ of facts
Create one (initially empty) set $S_{none}$ for facts that have
no constants in any example $e \in \mathcal{E}^+ \cup \mathcal{E}^-$
For each fact $f \in$ EDB Do
For each constant $c' \in f$ Do
$S_{c'} = S_{c'} \cup f$
If no set exists for $c$ then
$S_{none} = S_{none} \cup f$
Distribute the contents of $S_{none}$ among all constant sets
Determine load balance by summing all set cardinalities
to reflect total parallel EDB entries $K$
Define $min\_local\_load = \lceil K/p \rceil$
Distribute all sets $S_{c_i}$ evenly among the processors
so that each processor has an EDB of roughly
equal cardinality such that each node has EDB of
cardinality $\geq min\_local\_load$ as defined above.
*END ALGORITHM 5.1*

## 6 Global Hypergraph using Speculative Parallelism

In this parallelization, each Beowulf node searches the space of all possible rules independently and differently. All input files are the same on all machines. Therefore, each worker is discovering from the same background knowledge base. Every rule discovered by **INDED** is constructed by systematically appending chosen predicate expressions to an originally empty rule body. The predicate expressions are ranked by employing various algorithms, each of which designates a different search strategy. The highest ranked expressions are chosen to constitute the rule body under construction. In this parallelization of **INDED**, each node of the Beowulf employs a different ranking procedure, and hence, may construct very different rules.

We are considering two possibilities for handling the rules generated by each worker. In the first, as soon as a process converges (finds a valid set of rules), it broadcasts a message to announce the end of the procedure. When the message is received by other processes, they are terminated. The other possibility we

are considering is to combine the rules of each worker. Different processes may generate different rules due to the use of different ranking algorithms. These rules may be combined after all the processes are terminated, and only good rules are retained. In this way, not only can we speed up the mining process, but we can also achieve a better and richer quality of solutions.

## 7 Current Status and Results

The current status of our work in these parallelization schemes is as follows. We have implemented the naive decomposition and enjoyed a 50 per cent reduction in execution time. Thus far, however, the bulk of our efforts have centered on implementing and testing the data parallel implementation on an eleven node Beowulf cluster. Here, we also experienced a great reduction in execution time. Figure 1 illustrates the consistent reduction of time as the number of nodes increased. The problem domain with which we are currently experimenting relates to the diagnosis of diabetes. The accuracy of the discovered rules by the cluster has varied. The rule learned by serial **INDED** is

```
inject_insulin(A) <-- insulin_test4(A) .
inject_insulin(A) <-- iddm(A) .
```

We attribute the variance of rule accuracy by the clusters to our partitioning algorithm. We anticipate extensive refinement of this algorithm as we continue this work.

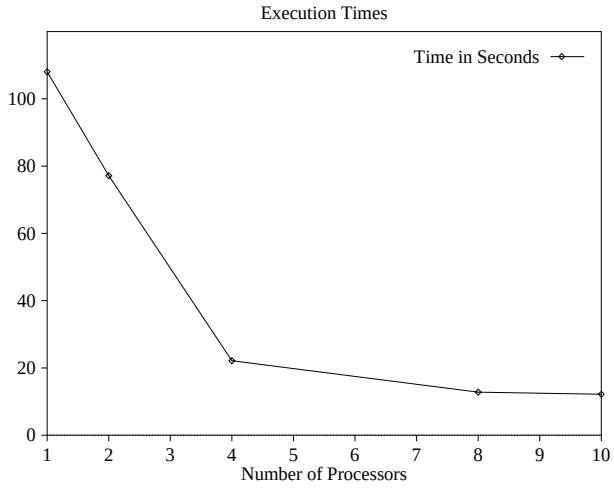



**Fig. 1.** Performance of Rule Mining on a Cluster.

## 8 Current and Future Work

We have delineated four parallelization schemes of transforming a large serial reasoning system that both deduces new facts as well as discovers new rules. In successfully implementing two of these schemes,we have found that one of the most interesting problems of parallel rule discovery is effective partitioning of the data. We have performed experimentation with one algorithm and anticipate extensive experimentation with new partitioning algorithms of the EDB and background knowlege. Additionally, we are currently implementing the speculative parallelization scheme discussed above, and are enhancing and devising new predicate ranking algorithms used by the induction engine.

## References


[AIS93] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Bulletin*, pages 207–216, May 1993.

[Buy99] Rajkumar Buyya. *High Performance Cluster Computing Programming and Applications*. Prentice-Hall, Inc, 1999.

[CHY96] M.S. Chen, J. Han, and P.S. Yu. Data mining: An overview from a database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), 1996.

[GL90] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080, 1990.

[GLS99] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI; Portable Parallel Programming with Message Passing Interface*. The MIT Press, 1999.

[KAK99] G Karypis,R. Agrawal, V. Kumar. Multilevel hypergraph partitioning: Applications in VLSI domain. *IEEE Transactions of VLSI Systems*, 7(1),pages 69–79, Mar 1999.

[LD94] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming*. Ellis Horwood, Inc., 1994.

[Mug92] Stephen Muggleton, editor. *Inductive Logic Programming*. Academic Press, Inc, 1992.

[PSF91] Piatetsky-Shapiro and Frawley, editors. *Knowledge Discovery in Databases*, chapter Knowledge Discovery in Databases: An Overview. AAAI Press/ The MIT Press, 1991.

[Qui86] J. R. Quindlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Sei99] Jennifer Seitzer. **INDED**: A symbiotic system of induction and deduction. In *MAICS-99 Proceedings Tenth Midwest Artificial Intelligence and Cognitive Science Conference*, pages 93–99. AAAI, 1999.

[SSC99] L. Shen, H. Shen, and L. Chen. New algorithms for efficient mining of association rules. In *7th IEEE Symp. on the Frontiers of Massively Parallel Computation, Annapolis, Maryland*, pages 234–241, Feb 1999.

[VRS91] A. VanGelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

[ZPO97] M. J. Zaki, S. Parthasarathy, and M. Ogihara. Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery*, 1:5–35, 1997.

# Parallelisation of C4.5 as a Particular Divide and Conquer Computation

Primo Becuzzi, Massimo Coppola, Salvatore Ruggieri, and Marco Vanneschi

Dipartimento di Informatica, Universita' di Pisa

**Abstract** In this work we show the research track and the current results about the application of structured parallel programming tools to develop scalable data-mining applications. We discuss the exploitation of the divide and conquer nature of the well known C4.5 classification algorithm in spite of its in-core memory requirements. The opportunity of applying external memory techniques to manage the data is advocated. Current results of the experiments are reported.

The main research goal of our group in the past years has been the design and implementation of parallel programming environments to ease the engineering of High Performance applications. To provide a test bed for the current environment, as well as to investigate the theoretical problems involved in the parallelisation of largely used algorithms, we are developing parallel versions of Data-Mining (DM) applications.

Work is ongoing [2,3] to develop DM computational kernels that exhibit both code and performance portability over various parallel architectures, where performance means parallel speed-up and scalability to large databases.

Here we present the current results about the C4.5 algorithm, focusing on two main issues:

- the parallel scalability achievable using structured programming tools (a software engineering perspective)
- the evaluation of strategies to improve the support for tree-structured irregular computations, regarding C4.5 as an algorithm of the Divide and Conquer class.

Dealing with data which exceeds in size the local memory is the most common issue in High Performance Data Mining. We plan to enhance C4.5, which is an in-core classifier, to efficiently manage huge data sets.

We will introduce in the SkIE programming environment some external memory operations (like scan and sort), that exploit at first the sum of all the local memories. We will use those primitives to turn the C4.5 algorithm into a kind of out-of-core classifier. We will investigate if the same approach can be applied to the lower level of the memory hierarchy, disk-resident data.

The the paper is organized as follows. In section 1 we define the problem. Section 2 describes our programming environment. We compare our approach with previous ones in section 3, together with the explanation of the first results. Section 4 explains in detail the current results. The aim and the improvements expected from future work are the subject of the last section.

## 1 Problem statement

We are interested in the parallelisation of the C4.5 core, that is the building of the decision tree, as described in [5]. We'll leave out the evaluation and simplification phases, and we won't discuss the replication of cases with unspecified attributes, nor the windowing and trials techniques, even if these can be conveniently applied.

General divide-and-conquer (*D&C*) algorithms split (*divide*) large problems into smaller and smaller ones of the same form; the smallest ones are solved, then a recomposition (*conquer*) phase combines the solutions of larger and larger subproblems, up to the initial one. We give here a *D&C* description of the core of C4.5.

**Input:** a database $D$ of $n$ elements, each one a $k$-tuple of attributes $(a_1, \ldots a_k)$. Each $a_i \in A_i$, where $A_i$ can be an interval (continuous attributes) or a finite set of values (categorical ones). One of the categorical attributes is distinguished as the class of each tuple.

**Output:** a decision tree $T$, i.e. a tree which predicts the class of tuples in terms of the values of the other attributes. Leaves of the tree are homogeneous subsets of the data. Each interior node (decision node) defines a split in the data determined by the test of the values of a single attribute, one branch is made for each outcome of the test.

**D1:** $\forall i \in \{1, k\}$ evaluate for attribute $i$ the information gain $g(D, i)$; let $j$ be the index that maximizes it.

**D2cat:** if $A_j$ is categorical, let $c = |A_j|$. Split $D$ into $c$ classes according to the value of $a_c$ of each tuple.

**D2cont:** if $A_j$ is continuous, let $c = 2$ and split $D$ in half such that $D_1 \leq D_2$. (*) Find *in all the data* the threshold value $t \in A_c$ that is the best approximation of the split point.

**D3:** build the root node for T and register the choices made so far.

**Rec:** each $D_1, \ldots D_c$ that does satisfy the stopping criterion becomes a leaf. Process recursively all the others.

**Conquer:** add each returned leaf or tree $T_1, \ldots T_c$ as the corresponding son of the root of $T$. Return $T$ as answer.

This is the *growth* phase of the tree, that is followed by a *prune* phase which is less hard to accomplish and we do not describe here.

Steps D1–D3 are the divide phase. The cost of step D1 is $O(n)$ operations for each categorical attribute, $O(n \log n)$ for continuous ones. Categorical attributes already used by an ancestor node are never checked again, since their information gain is zero. Both D2 steps, which are exclusive, require $O(n)$ operations, but finding the threshold (*) in step D2cont has an additional cost $O(N \log N)$ in the size $N$ of the whole initial database.

This behaviour, more deeply analyzed in [6], can impair load predictability and balancing of parallel implementations, and prevents them from being truly *D&C* computations. As already noted in [4], the threshold information is not

needed until the pruning phase, so the threshold selection can be delayed and computed in an amortized way at the end of the classification phase.

We will assume now that the Conquer step in C4.5 has nearly no cost, and we will come back later to this point.

From an I/O operational point of view, all the steps require linear scans, or sorting and searching through the data in the current partition. Other parallel algorithms like [7] and [8] have been devised to deal with huge data partitions through special data structures and scheduling policies. We want to design a small set of general primitives that allow us to express the algorithm and can be implemented in a standard, user-friendly way in a high-level language. The research in the field of external memory algorithms [10] has produced theoretical analysis and scalable solutions for such a class of basic operations, that are suitable for application to the memory hierarchy of a parallel architecture.

We wanted to maintain the C4.5 sequential results as a reference point[1], and exploit the *D&C* aspect of the computation. Thus solutions that require to change the split criterion to binary, like in [7], or using a different splitting method [8], were regarded as not fully satisfying.

## 2 The Programming Environment

The SkIE environment [9] we are using and developing is a parallel coordination language based on the concept of *skeleton* . A skeleton is a basic form of parallelism that builds blocks of parallel code by composing simpler blocks (eventually sequential code) in an abstract, modular way. The aim of this approach is at reaching both source code and performance portability across different parallel architectures.

The skeleton run-time support deals with almost all the low level details of parallelism and concurrency (i.e. process mappings, communications, scheduling and load balancing). Application development is thus enhanced by software reuse, rapid prototyping, and a lesser need for performance debugging.

The SkIE user has to pick up a conceptual parallelisation that can be expressed using only the available skeletons. He is relieved of most of the low-level details of the parallelisation, but on the other hand he has little intervention on these aspects, in a trade-off between expressive power and efficiency.

The SkIE semantics is data-flow oriented, with an explicit vision of the streams of tasks. Among the used skeletons, the farm exploits task parallelism over a stream, providing automatic load distribution and balancing. The pipe is the functional composition, with pipeline parallelism exploited. A loop skeleton allows repeated processing of (part of) a stream by another parallel module. Other skeletons like map, reduce, deal with the basic forms of data-parallelism.

All communication set-up is transparently handled by the compiler at the interfaces between the modules, but this imposes some constraints on having fixed-size data structures as parameters. To overcome this limitation and allow

[1] Even if it is argued in the literature that a split criterion that requires sorting uses too much computation w.r. to the accuracy of the results.

more expressiveness, a virtual shared memory support is being integrated into the high-level interface. The abstraction is defined of dynamic, out-of-core shared data objects (SO), that are stored in the aggregate memory of the computer.

## 3 Related work and first experiments

With respect to the parallelisation, following [4] we can classify most of the previous approaches into *attribute parallel*, which assign each $A_i$ to a processor to execute step D1 in parallel, *data parallel* ones, which split the database among the processors and handle most operations collectively, and *task parallel* ones, which try to exploit the recursive definition to start separate computations at each branch.

The tree structure can be highly irregular, and this reflects in the computation. Since the classification workload cannot be foreseen for any given subtree, pure data parallel solutions in the literature exhibit no good parallel scalability, like the synchronous approach in [8]. As discussed in the same paper, even partitioned, more task-oriented solutions with static load-balancing cannot properly handle the irregular load, and a hybrid solutions is proposed.

We choose, instead, to explore task parallelism at first. A similar choice was made in [11], where data parallelism is combined with pipelining. Our experiments are a valuable research alternative, since we take advantage of the automatic load balancing and task pipelining in the SklE skeletons.

Our prototypes exploits task parallelism, with each worker processor expanding a node of the C4.5 tree into a subtree, up to a certain depth $l$.

We refer to [1] for a detailed presentation of the first parallel version, which uses farm parallelism in a structure close to that in Fig.1.

Since the user has no control on the task distribution done by the farm skeleton, in our pure task approach there are only two parameters to tune: the depth $l$ of each expansion, and the selection order of waiting tasks.

Using a fixed, on-demand scheduling of tasks requires some property to hold for the given task stream. This first version of the program, which we call MP, was impaired by an excessive communication load and computation variance. The communications were all the same size, comparable to the database size.

The computation of thresholds in step D2cont (*), and the use of too high values for the $l$ parameter led to a highly variable worker load, which resulted in poor load balancing. Only minor improvements were obtained by using more complex or adaptive strategies to adjust the value of $l$ at run-time.

To improve upon the MP solution, we have started a new research path by introducing in the environment the abstraction of shared objects (SO), to be used to remove data replication and unwanted centralization points. This strategy has lead to (1) use the SO to improve communications, (2) switch to a full *D&C* computation, delaying threshold calculation. Next steps are (3) distribute the database among the workers, providing remote access methods, (4) turn the decision tree into a SO itself, thus removing the centralization point.

It is correctly pointed out in [8] that task parallelism alone for classification is not scalable because of large nodes. A further step will be to introduce data-parallel collective operations on the external data structures. Once distributed operation are provided, a first phase of the computation could proceed in a data parallel fashion (either doing attribute or data partitioning).

We argue that it is possible to achieve an efficient implementation of these operations. We will take advantage of the fact that, in the SkIE environment, this implementation is completely independent from the details of the DM algorithms.

## 4 Current results

Up to now, the path described has been followed to its 2nd step, with work ongoing to reach step 3. We call the prototype at step 1 MP+SM, and the current one DT (it is the same with the Delayed Threshold calculation ).

Fig. 1 shows the parallel structure we used for both. The data are still replicated. The task parallelism is applied in the Divide phase through a farm skeleton, each task contains a single node which has to be expanded into a subtree. As we have said, the depth $l$ of the computed subtrees is used to tune the amount of expansion. The expansion politics is explained later.

A single process owns the decision tree and does the Conquer step; it collects and joins subtrees. Nodes still needing to be expanded are sent back to the input of the parallel loop. The Conquer phase is executed in pipeline with the Divide phase inside the loop, to hide its latency.

All the test results are measured on a QSW CS-2 with 10 SPARC processors and a high performance communication network, using the data set Adult from the UCI repository.

With the farm dynamic load balancing, we don't need to evaluate in advance the computation needed by the subtree, which is repeatedly expanded no more than $l$ levels each time. It is enough that the variance of the node workload is bounded. Our work has been oriented at reducing this variance.

Each task requires an amount of communication proportional to the size of its partition, so using the shared objects to store the data keeps communication and computation costs closer. The gain is clear in Fig.2a from the comparison between MP and MP+SM execution times.

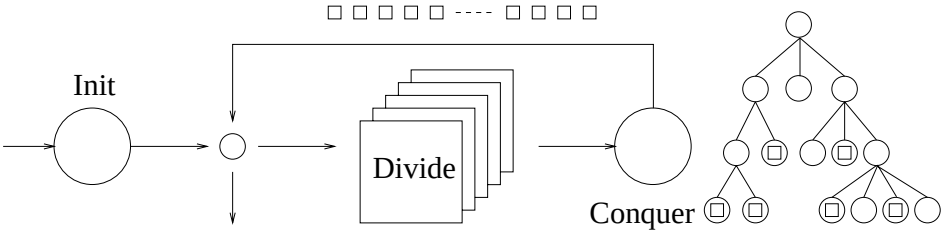



**Figure1.** Parallel structure of the prototype

The communication latency becomes almost independent from the size of the task, see in Fig.3b the communication cost for one worker. The communication overhead exceeds the computation only for very small tasks.

The MP+SM solution has another source of load imbalancing in the threshold calculation, which is $O(N \log N)$ operations. Delaying the calculation of thresholds until the end of the growth phase minimizes the load variance, allows a *D&C* formalisation and reduces the computational effort [4,6].

The comparison of MP+SM and TD in fig.3a underlines the positive effect of delaying the threshold calculation. The overall effect on the completion time is even greater, as reported in fig.2a.

Now the behaviour of the application is easier to analyze than with MP and MP+SM, with different parameters having more understandable effects.

Using a simple depth-first visit does not allow to exploit task parallelism in full. Moreover, since the average expansion cost increases with node size, giving priority to the bigger nodes results in a regular, decreasing computational load, which can be easily balanced.

The depth first selection in the first versions of MP made unsuccessful both the standard scheduling policies of the support and any attempt to use adaptive expansion at the workers.

If task computation time is too low w.r. to communication, parallelism is useless. Varying the amount of node expansion in the sequential code now succeeds in controlling the computation to communication ratio. In fig.3b it is easy to see the task size where communication (and idle) time and computation time on average balance. Here a second parameter controlling the expansion comes into play, the maximum amount of nodes for a subtree. In fig.2b we can see that raising this parameter from 512 to 2048 allows most of the small nodes to be computed immediatly instead of becoming new smaller tasks, this way improving the computation to communication ratio.

In fig.3b the communication time is higher for the same code when run with 6 workers instead of just one. The Conquer process is too busy to keep up with the output of the farm: it definitely becomes a bottleneck. This is partly due to the amount of communication, and mostly to the fact that the sequential code inside the module spends most of the time inside recursive visits the tree.

Theoretically the conquer operation should be fast, but the amount of work required to update the centralized data structure, as well as some inefficiency in its implementation, are no longer negligible. The effect shows up clearly with more and more worker nodes, and gets worse when there is more useful parallelism, as the arrival rate of new tasks increases.

## 5 Expected Results and Future work

The next change to the application will be needed to remove the Conquer bottleneck. We could change the tree visits into heap accesses, lowering the overhead, but we see as a long term solution to store the tree in the shared memory space.

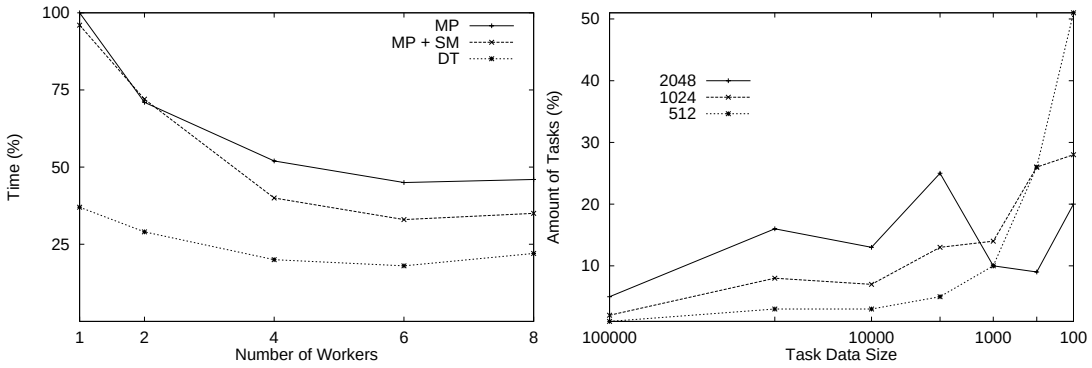



**Figure2.** (a) Results of the three different parallel implementations. (b) Distribution of task size with a varying subtree buffer size.

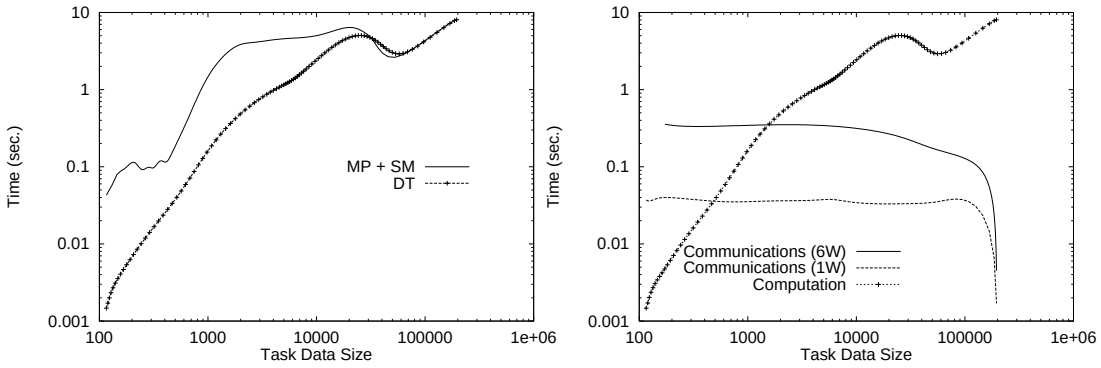



**Figure3.** (a) Computation time of task w.r. to size, with and without threshold calculation. (b) Computation vs communication and idle time per task, 1 and 6 workers.

Handling the merge operations in a decentralized way will lead to higher parallel scalability.

The utilization of the shared space is also needed to overcome the current assumption that the database fits in memory. A separate development path has already started about maintaining the database and the tree as linked structures in the shared memory space.

The current results, obtained through the integration of the shared objects abstraction into the SkIE skeleton programming model, strongly suggest that the approach is effective in solving the communication/computation problems for a class of irregular computations involving large data sets.

SkIE makes also easy to use the current parallel application as a building block for bigger ones. It is straightforward to set up a control loop that uses a set of C4.5 parallel blocks to scan multiple windows of a database at the same time.

We have not yet addressed the first few iterations in the task parallelisation: the expansion time for the first nodes accounts for 10% to 20% of the computation time, and things would get worse with bigger databases. The solution will be to compute the huge tasks exploiting data parallelism, and to switch to task parallelism as soon as no big tasks are still waiting. This can be done by using a new skeleton of the SkIE environment, which can switch from on demand scheduling to data-driven one. Secondary memory algorithms could be used to efficiently implement global operations like distributed sorting of the data.

Summing up we could move from a main-memory implementation of C4.5 to a skeleton structured implementation, where the main layer of the memory is the aggregate memory of the parallel architecture, whether a massively parallel architecture or a cluster of workstation with virtual shared memory support. The solutions devised should also be applied to the design of a generic skeleton composition that implements, at least, those D&C computations in which the conquer function is basically a merge operation.

## References


1. P. Becuzzi, M. Coppola, D. Laforenza, S. Ruggieri, D. Talia, and M. Vanneschi. Data analysis and data mining with parallel architectures: Techniques and experiments. Technical report, Consorzio Pisa Ricerche, project "Parallel Intelligent Systems for Tax Fraud Detection", December 1998.
2. P. Becuzzi, M. Coppola, and M. Vanneschi. Association rules in large databases, additional results. http://www.di.unipi.it/~coppola/ep99talk.ps, Aug 1999.
3. P. Becuzzi, M. Coppola, and M. Vanneschi. Mining of Association Rules in Very Large Databases: a Structured Parallel Approach. In *Euro-Par'99 Parallel Processing*, volume 1685 of *LNCS*. Springer, 1999.
4. John Darlington, Yike Guo, Janjao Sutiwaraphun, and Hing Wing To. Parallel Induction Algorithms for Data Mining. In *Advances in intelligent data analysis: reasoning about data IDA'97*, volume 1280 of *LNCS*, 1997.
5. J.R. Quinlan. *C 4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, 1993.
6. S. Ruggieri. Efficient C4.5. Draft, http://www-kdd.di.unipi.it/software.
7. John Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Proceedings of the 22nd VLDB Conference*, 1996.
8. A. Srivastava, E.H. Han, V. Kumar, and V. Singh. Parallel Formulations of Decision-Tree Classification Algorithms. *Data Mining and Knowledge Discovery*, 3(3), 1999.
9. M. Vanneschi. PQE2000: HPC Tools for Industrial Applications. *IEEE Concurrency: Parallel, Distributed & Mobile Computing*, 6(4):68–73, Oct-Dec 1998.
10. Jeffrey Scott Vitter. External Memory Algorithms and Data Structures: Dealing with MASSIVE DATA. Draft, http://www.cs.duke.edu/~jsv, January 2000.
11. Mohammed J. Zaki, Ching-Tien Ho, and Rakesh Agrawal. Scalable Parallel Classification for Data Mining on Shared-Memory Multiprocessors. In *Proc. of the IEEE Int'l Conference on Data Engineering*, March 1999.

# Scalable Parallel Clustering for Data Mining on Multicomputers

D. Foti, D. Lipari, C. Pizzuti and D. Talia

ISI-CNR
c/o DEIS, UNICAL
87036 Rende (CS), Italy
{pizzuti,talia}@si.deis.unical.it

**Abstract.** This paper describes the design and implementation on MIMD parallel machines of *P-AutoClass*, a parallel version of the *AutoClass* system based upon the Bayesian method for determining optimal classes in large datasets. The *P-AutoClass* implementation divides the clustering task among the processors of a multicomputer so that they work on their own partition and exchange their intermediate results. The system architecture, its implementation and experimental performance results on different processor numbers and dataset sizes are presented and discussed. In particular, efficiency and scalability of *P-AutoClass* versus the sequential *AutoClass* system are evaluated and compared.

## 1 Introduction

Clustering algorithms arranges data items into several groups so that similar items fall into the same group. This is done without any suggestion from an external supervisor, classes and training examples are not given a priori. Most of the early cluster analysis algorithms come from the area of statistics and have been originally designed for relatively small data sets. In the recent years, clustering algorithms have been extended to efficiently work for knowledge discovery in large databases and some of them are able to deal with high-dimensional feature items. When used to classify large data sets, clustering algorithms are very computing demanding and require high-performance machines to get results in reasonable time. Experiences of clustering algorithms taking from one week to about 20 days of computation time on sequential machines are not rare. Thus, scalable parallel computers can provide the appropriate setting where to execute clustering algorithms for extracting knowledge from large-scale data repositories.

Recently there has been an increasing interest in parallel implementations of data clustering algorithms. Parallel approaches to clustering can be found in [8, 4, 9, 5, 10]. In this paper we consider a parallel clustering algorithm based on Bayesian classification for distributed memory multicomputers. We propose a parallel implementation of the AutoClass algorithm, called *P-AutoClass*, and validate by experimental measurements the scalability of our parallelization strategy. The *P-AutoClass* algorithm divides the clustering task among the processors of a parallel

machine that work on their own partition and exchange their intermediate results. It is based on the message passing model for shared-nothing MIMD computers.

This paper describes the design and implementation of *P-AutoClass*. Furthermore, experimental performance results on different processor numbers and dataset sizes are presented and discussed. The rest of the paper is organized as follows. Section 2 provides a very short overview of Bayesian classification and sequential AutoClass. Section 3 describes the design and implementation of *P-Autoclass* for multicomputers. Section 4 presents the main experimental performance results of the algorithm. Section 5 describes related work and section 6 contains conclusions.

## 2 Bayesian Classification and AutoClass

The Bayesian approach to unsupervised classification provides a probabilistic approach to induction. Given a set $X = \{X_1, \ldots, X_I\}$ of data instances $X_i$, with unknown classes, the goal of Bayesian classification is to search for the best class description that predict the data. Instances $X_i$ are represented as ordered vectors of attribute values $\overrightarrow{X_i} = \{X_{i1}, \ldots, Xik\}$.

In this approach class membership is expressed probabilistically, that is an instance is not assigned to a unique class, instead it has a probability of belonging to each of the possible classes. The classes provides probabilities for all attribute values of each instance. Class membership probabilities are then determined by combining all these probabilities. Class membership probabilities of each instance must sum to 1, thus there not precise boundaries for classes: every instance must be a member of some class, even though we do not know which one. When every instance has a probability of about 0.5 in any class, the classification is not well defined because it means that classes are abundantly overlapped. On the contrary, when the probability of each instance is about 0.99 in its most probable class, the classes are well separated.

A Bayesian classification model consists of two sets of parameters: a set of discrete parameters *T* which describes the functional form of the model, such as number of classes and whether attributes are correlated, and a set of continuos parameters $\vec{V}$ that specifies values for the variables appearing in *T*, needed to complete the general form of the model. The probability of observing an instance having particular attribute value vector is referred to as *probability distribution or density function* (p.d.f.). Given a set of data *X*, AutoClass searches for the most probable pair $\vec{V}$, *T* which classifies *X*. This is done in two steps:

- For a given *T*, AutoClass seeks the maximum posterior (MAP) parameter values $\vec{V}$.
- Regardless of any $\vec{V}$, AutoClass searches for the most probable *T*, from a set of possible *Ts* with different attribute dependencies and class structure.

Thus there are two levels of search: *parameter level search* and *model level search*. Fixed the number classes and their class model, the space of allowed parameter values is searched for finding the most probable $\vec{V}$. Given the parameter values, AutoClass

calculates the likelihood of each case belonging to each class *L* and then calculates a set of weights $w_{ij}=(L_i / \Sigma_j L_j)$ for each case. Given these weigths, weighted statistics relevant to each term of the class likelihood are calculated. These statistics are then used to generate new MAP values for the parameters and the cycle is repeated.

Based on this theory, Cheeseman and colleagues at NASA Ames Research Center developed AutoClass [1] originally in Lisp. Then the system has been ported from Lisp to C. The C version of AutoClass improved the performance of the system of about ten times and has provided a version of the system that can be easily accessed and used by researchers at a variety of universities and research laboratories.

## 3 P-AutoClass

In spite of the significant improvement of the C version performance, because of the computational needs of the algorithm, the execution of AutoClass with large datasets requires times that in many cases are very high. For instance, the sequential AutoClass runs on a dataset of 14K tuples, each one composed of a few hundreds bytes, have taken more the 3 hours on Pentium-based PC. Considering that the execution time increases linearly with the size of dataset, more than 1 day is necessary to analyze a dataset composed of about 140K tuples, that is not a very large dataset. For the clustering of a satellite image AutoClass took more than 130 hours [6] and the analysis of protein sequences the discovery process required from 300 to 400 hours [3].

These considerations and experiences suggest that it is necessary to implement faster versions of AutoClass to handle very large data set in reasonable time. This can be done by exploiting the inherent parallelism present in the AutoClass algorithm implementing it in parallel on MIMD multicomputers. Among the different parallelization strategies, we selected the SPMD approach. In AutoClass, the SPMD approach can be exploited by dividing up the dataset among the processors and by the parallel updating on different processors of the weights and parameters of classifications. This strategy does not require to replicate the entire dataset on each processor. Furthermore, it also does not have load balancing problems because each processor execute the same code on data of equal size. Finally, the amount of data exchanged among the processors is not so large since most operations are performed locally at each processor.

### 3.1 Design of the parallel algorithm

The main steps of the structure of the AutoClass program are described in figure 1. After program starting and structure initialization, the main part of the algorithm is devoted to classification generation and evaluation (*step 3*). This loop is composed of a set of substeps specified in figure 2. Among those substeps, the *new try of classification* step is the most computationally intensive. It computes the weights of each items for each class and computes the parameters of the classification. These operations are executed by the function **`base_cycle`** which calls the three functions **`update_wts`**, **`update_parameters`** and **`update_approximations`** as shown in figure 3.

1. Program Start and Files Reading ;
2. Data Structures Initialized ;
3. Classification Generation and Evaluation (called also *BIG_LOOP*) ;
4. Check the Stopping Conditions, if they are not verified go to step **3**, else go to step **5**;
5. Store Results on the Output Files.

**Fig. 1.** Scheme of the sequential AutoClass algorithm.

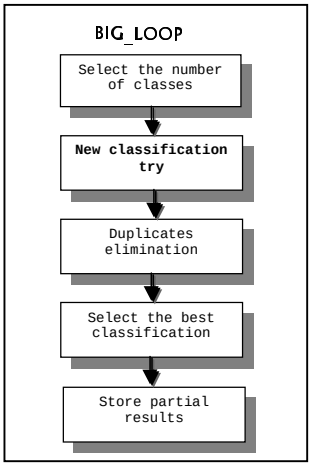



**Fig. 2.** Main steps of the *BIG_LOOP*.

We analyzed the time spent in the **`base_cycle`** function and it resulted about the 99,5% of the total time, therefore we identified this function as that one where parallelism must be exploited to speed up the AutoClass performance.

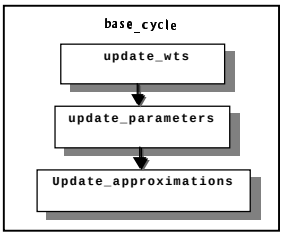



**Fig. 3.** The structure of the *base_cycle* function.

In particular, analyzing the time spent in each of the three functions called by **`base_cycle`**, it appears, as observed in other experiences [7], that the **`update_wts`** and **`update_parameters`** functions are the time consuming functions whereas the time spent in the **`update_approximation`** is negligible. Therefore, we studied the parallelization of these two functions using the SPMD approach. To maintain the same semantics of the sequential algorithm of AutoClass, we designed the parallel

version by partitioning data and local computation on each of **P** processors of a distributed memory MIMD computer and by exchanging among the processors all the local variables that contribute to form global values of a classification.

### 3.1.1 Parallel update_wts

In AutoClass the class membership of each data item is expressed probabilistically. Thus every item has a probability that it belongs to each of the possible classes. In the AutoClass algorithm, class membership is expressed by weights. The function **`update_wts`** calculates the weights $w_{ij}$ for each item $i$ of the active classes to be the normalized class probabilities with respect to the current parameterizations.

The parallel version of this function first calculates on each processing element the weights $w_{ij}$ for each item belonging to the local partition of the data set and sum the weights $w_j$ of each class $j$ ($w_j=\Sigma_i\ w_{ij}$) relatively to its own data. Then all the partial $w_j$ values are exchanged among all the processors and summed in each of them to have the same value in every processor. This strategy is described in figure 4.

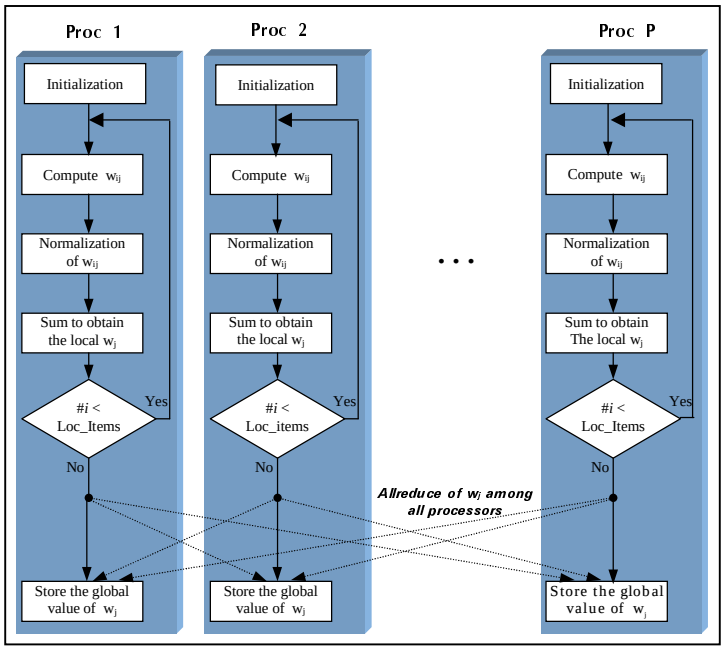



**Fig. 4.** The parallel version of the *update_wts* function.

To implement the total exchange of the $w_j$ values in the **`update_wts`** function we used a *global reduction* operation (`Allreduce`) that sums all the local copies in the all processes (*reduction* operation) and places the results on all the processors (*broadcast* operation).

### 3.1.2 Parallel update_parameters

The **`update_parameters`** function computes for each class a set of class posterior parameter values, which specify how the class is distributed along the various

attributes. To do this, the function is composed of three nested loops, the external loop scans all the classes, then for each class all the attributes are analyzed and in the inner loop all the items are read and their values are used to compute the class parameters.

In parallelizing this function, we executed the partial computation of parameters in parallel on all the processors, then all the local values are collected on each processor before to utilize them for computing the global values of the classification parameters. Figure 5 shows the scheme of the parallel version of the function. To implement the total exchange of the parameter values in the **`update_parameters`** function we used a *global reduction* operation that sums all the local copies in all the processes and places the results on every processor.

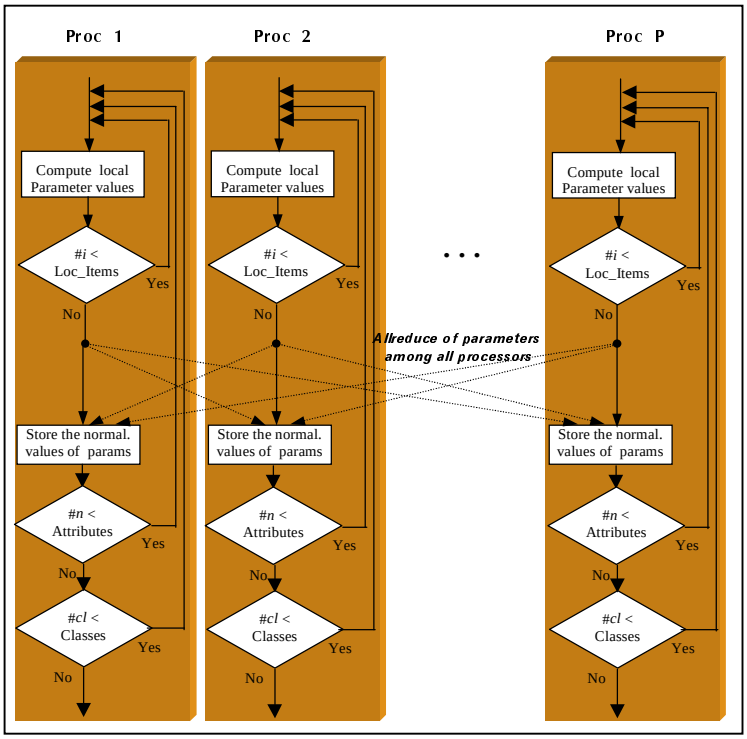



**Fig. 5.** The parallel version of the *update_parameters* function.

*P-AutoClass* has been implemented using the Message Passing Interface (MPI) toolkit on a Meiko Computing Surface 2 using the version 3.3 of sequential AutoClass C. Because of the large availability of MPI, *P-AutoClass* is portable practically on every parallel machine from supercomputers to PC clusters.

## 4 Experimental results

We run our experiments on a Meiko CS 2 with up to 10 SPARC processors connected by a fat tree topology with a communication bandwidth of 50 Mbytes/s in both

directions. We used a synthetic dataset composed of 100000 tuples each one composed of two real attributes. To perform our experiments we used different partitions of data from 5000 tuples to the entire dataset, and asked the system to find the best clustering starting with different number of clusters (*start_j_list=2, 4, 8, 16, 24, 50, 64*). Each classification has been repeated 10 times and results presented here represent the mean values obtained after these classifications. We measured the elapsed time, the speedup and the scaleup of P-AutoClass. *Speedup* gives the efficiency of the parallel algorithm when the number of processors varies. It is defined as the ratio of the execution time for clustering a dataset on *1* processor to the execution time for clustering the same dataset on *P* processors. Another interesting measure is scaleup. *Scaleup* captures how a parallel algorithm handles larger datasets when more processors are available.

Figure 6 shows the elapsed times of P-AutoClass on different numbers of processors. We can see that the total execution time substantially decreases as the number of used processors increases. In particular, for the largest datasets the time decreases in a more significant way. We can observe that as the dataset size increases the time gain increases as well.

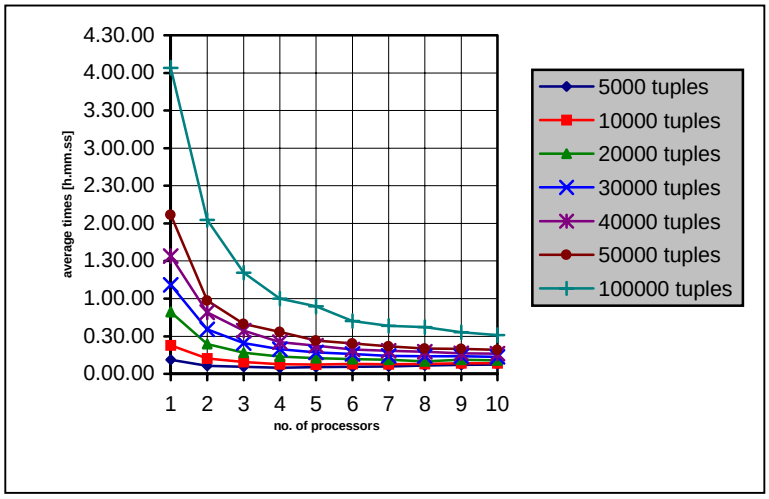



**Fig. 6.** Average elapsed times of *P-AutoClass* on different numbers of processors.

In figure 7 the speedup results obtained for different datasets are given. We can observe that the *P-AutoClass* algorithm scales well up to 10 processors for the largest datasets, whereas for small datasets the speedup increases until the optimal number of processors are used for the given problem (e.g., 4 procs for 5000 tuples or 8 procs for 20000 tuples). When more processes are used we observe that the algorithm does not scale because the processors are not effectively used and the communication costs increases.

For scaleup measures we evaluated the execution time of a single iteration of the *base_cycle* function by keeping the number of data items per processor fixed while increasing the number of processors. To obtain more stable results we asked *P-AutoClass* to group data into 8 and 16 clusters. Figure 8 shows the scaleup results. For all the experiments we have 10000 tuples per processor. We started with 10000 tuples

on 1 processor up to 100000 tuples on 10 processors. It can be seen that the parallel AutoClass algorithm, for a given classification, shows a nearly stable pattern. Thus it delivers nearly constant execution times in number of processors showing good scaleup.

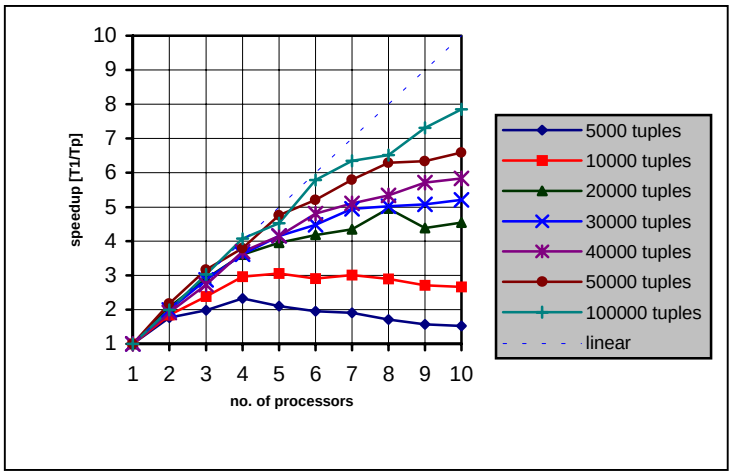



**Fig. 7.** Speedup of *P-AutoClass* on different numbers of processors.

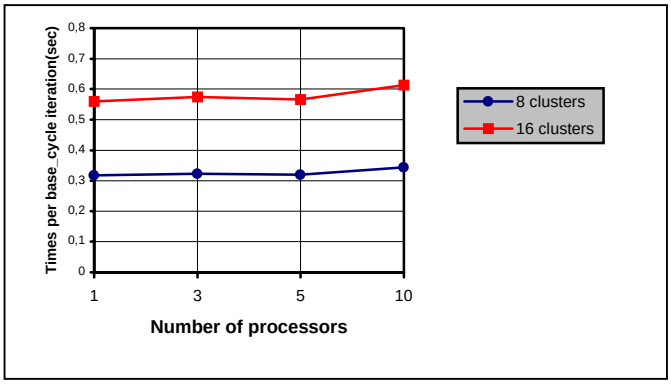



**Fig. 8.** Scaleup of the *base_cycle* of *P-AutoClass* on different sets of tuples scaled by the number of processors.

## 5 Related work

In the past few years there has been an increasing interest in parallel implementations of data mining algorithms [2]. The first approaches to the parallelization of AutoClass have been done on SIMD parallel machines by using compilers that automatically generated data-parallel code starting from the sequential program to which only few instructions have been added. The first data-parallel version of AutoClass has been developed in *Lisp to run on a Connection Machine-2 [1], and the second one has been developed adding C* code to the C source to run it on a CM-5 [9]. These

approaches are simple from the programming point-of-view but they do not exploit all the potential parallelism. In fact, it is not known how well compilers extract parallelism from sequential programs.

The only experience we know about the implementation of AutoClass on a MIMD computer is described in [7]. This prototype is based on the exploitation of parallelism in the `update_wts` function. Concerning to the SIMD approach, our implementation is more general and allows to exploit parallelism in a more complete, flexible, and portable way. On the other hand, considering the mentioned MIMD parallel implementation, *P-AutoClass* exploits parallelism also in the parameters computing phase, with a further improvement of performance.

## 6 Conclusion

In this paper we proposed *P-AutoClass*, a parallel implementation of the AutoClass algorithm based upon the Bayesian method for determining optimal classes in large datasets. We have described and evaluated the *P-AutoClass* algorithm on a MIMD parallel computer. The experimental results show that *P-AutoClass* is scalable both in terms of speedup and scaleup. This means that for a given dataset, the execution times can be reduced as the number of processors increases, and the execution times do not increase if, while increasing the size of datasets, more processors are available. Finally, our algorithm is easily portable to various MIMD distributed-memory parallel computers that are now currently available from a large number of vendors. It allows to perform efficient clustering on very large datasets significantly reducing the computation times on several parallel computing platforms.

## References


1. P. Cheeseman and J. Stutz. Bayesian Classification (AutoClass): Theory and Results. In *Advances in Knowledge Discovery and Data Mining*, AAAI Press/MIT Press, pp. 61-83, 1996.
2. A.A. Freitas and S.H. Lavington. *Mining Very Large Databases with Parallel Processing*. Kluwer Academic Publishers, 1998.
3. L. Hunter and D.J. States. Bayesian Classification of Protein Structure. *IEEE Expert*, 7(4):67-75, 1992.
4. D. Judd, P. McKinley, and A. Jain. Large-Scale Parallel Data Clustering. *Proceedings of the Int. Conf. on Pattern Recognition*, 1996.
5. D. Judd, P. McKinley, A. Jain. Performance Evaluation on Large-Scale Parallel Clustering in NOW Environments. *Proceedings of the Eight SIAM Conf. on Parallel Processing for Scientific Computing*, Minneapolis, March 1997.
6. B. Kanefsky, J. Stutz, P. Cheeseman, and W. Taylor. An Improved Automatic Classification of a Landsat/TM Image from Kansas (FIFE). Technical Report FIA-94-01, NASA Ames Research Center, May 1994.
7. R. Miller and Y. Guo. Parallelisation of AutoClass. *Proceedings of the Parallel Computing Workshop (PCW'97)*, Canberra, Australia, 1997.
8. C.F. Olson. Parallel Algorithms for Hierarchical Clustering. *Parallel Computing*, 21:1313-1325, 1995.
9. J.T. Potts. Seeking Parallelism in Discovery Programs. Master Thesis, University of Texas at Arlington, 1996.
10. K. Stoffel and A. Belkoniene. Parallel K-Means Clustering for Large Data Sets. *Proceedings Euro-Par '99*, LNCS 1685, pp. 1451-1454, 1999.

# Exploiting Dataset Similarity for Distributed Mining *

Srinivasan Parthasarathy and Mitsunori Ogihara

Department of Computer Science
University of Rochester
Rochester, NY 14627–0226

{srini,ogihara}@cs.rochester.edu

**Abstract.** The notion of similarity is an important one in data mining. It can be used to provide useful structural information on data as well as enable clustering. In this paper we present an elegant method for measuring the similarity between homogeneous datasets. The algorithm presented is efficient in storage and scale, has the ability to adjust to time constraints. and can provide the user with likely causes of similarity or dis-similarity.
One potential application of our similarity measure is in the distributed data mining domain. Using the notion of similarity across databases as a distance metric one can generate clusters of similar datasets. Once similar datasets are clustered, each cluster can be independently mined to generate the appropriate rules for a given cluster. The similarity measure is evaluated on a dataset from the Census Bureau, and synthetic datasets from IBM.

## 1 Introduction

Similarity is a central concept in data mining. Research in this area has primarily progressed along two fronts: object similarity [2, 8, 7] and attribute similarity [5, 9]. The former quantifies how far from each other two objects in the database are while the latter refers to the distance between attributes. Discovering the similarity between objects and attributes enables reduction in dimensions of object profiles as well as provides useful structural information on the hierarchy of attributes.

In this paper we extend this notion of similarity to homogeneous distributed datasets. Discovering the similarity between datasets enables us to perform "meaningful" distributed datamining. Large business organizations with nation-wide and international interests usually rely on a homogeneous distributed database to store their transaction data. This leads to multiple data sources with a common structure. In order to analyze such collection of databases it seems important to cluster them into small number of groups to contrast global trends with local trends rather than apply traditional methods which simply combine them into a single logical resource. A limitation of traditional methods is that the joining is not based on the database characteristics, such as the demographic, economic conditions, and geo-thermal conditions. Mining each database individually is unacceptable as it is likely to generate too many spurious patterns (outliers). We argue for a hybrid solution. First cluster

* This work was supported in part by NSF grants CDA–9401142, CCR–9702466, CCR–9701911, CCR–9725021, INT–9726724, and CCR–9705594; and an external research grant from Digital Equipment Corporation.

the datasets, and then apply the *traditional distributed mining* approach to generate a set of rules for each resulting cluster.

The primary problem with clustering such homogenous datasets is to identify a suitable distance (similarity) metric. The similarity metric depends not only on the kind of mining task being performed but also on the data. Therefore, any measure of similarity should be flexible to both the needs of the task, and data. In this paper we present and evaluate such a similarity metric for distributed association mining. We believe that this metric can be naturally extended to handle other mining tasks such as discretization and sequence mining as well. We then show how one can cluster the database sources based on our similarity metric in an I/O and communication efficient manner. A novelty of our approach to clustering, other than the similarity measure is how we merge datasets without communicating the raw data itself.

The rest of this paper is organized as follows: Section 2 formally defines the problem, and describes our proposed similarity measure. We then present our method for clustering distributed datasets using the aforementioned similarity metric in Section 3. We experimentally validate our approach on real and synthetic datasets in Section 4. Finally, we conclude in Section 5.

## 2 Similarity Measure

Our similarity measure adopts an idea recently proposed by Das et al [5] for measuring attribute similarity in transaction databases. They propose comparing the attributes in terms of how they are individually correlated with other attributes in the database. The choice of the other attributes (called the probe set) reflects the examiner's viewpoint of relevant attributes to the two. A crucial issue in using this similarity metric is the selection of the probe set. Das *et al.* [5] observed that this choice strongly affects the outcome. However, they do not provide any insight to automating this choice when no apriori knowledge about the data is available. Furthermore, while the approach itself does not limit probe elements to singleton attributes, allowing for complex (boolean) probe elements and computing the similarities across such elements can quickly lead to problems of scale.

We propose to extend this notion of similarity to datasets in the following manner. Our similarity measure compares the datasets in terms of how they are correlated with the attributes in the database. By restricting ourselves to frequently occurring patterns, as probe elements, we can leverage existing solutions (Apriori [3]) for such problems to generate and interactively prune the probe set. This allows us to leverage certain powerful features of associations to handle the limitations described above. First, by using associations as the initial probe set we are able to obtain a "first guess" as to the similarity between two attributes. Second, since efficient solutions for the association problem exist, similarities can be computed rapidly. Third, once this "first guess" is obtained we are able to leverage and extend [1] existing work in interactive (online) association mining [1] to quickly compute similarities under boolean constraints, provide insights into the causes of similarity and dis-similarity, as well as to allow the user to interact and prune the probe space. Finally, we can leverage existing work on sampling to compute the similarity metric accurately and efficiently in a distributed setting.

[1] In addition to the interactions supported in [1] we also support **influential attribute** identification. This interaction basically identifies the (set of) probe attribute(s) that contribute most to the similarity metric.

### 2.1 Association Mining Concepts

We first provide basic concepts for association mining, following the work of Agrawal *et al.* [3]. Let $\mathcal{I} = \{i_1, i_2, \cdots, i_m\}$ be a set of $m$ distinct *attributes*, also called *items*. A set of items is called an *itemset* where for each nonnegative integer $k$, an itemset with exactly $k$ items is called a *k-itemset*. A *transaction* is a set of items that has a unique identifier *TID*. The *support* of an itemset $A$ in database $\mathcal{D}$, denoted $\sup_{\mathcal{D}}(A)$, is the percentage of the transactions in in $\mathcal{D}$ containing $A$ as the subset. The itemsets that meet a user specified *minimum support* are referred to as *frequent* itemsets or as *associations*. An *association rule* is an expression of the form $A \Rightarrow B$, where $A$ and $B$ are disjoint itemsets. The *confidence* of an association rule $A \Rightarrow B$ is $\frac{\sup_{\mathcal{D}}(A \cup B)}{\sup_{\mathcal{D}}(A)}$, i.e., the fraction of the datasets containing $B$ over those containing $A$.

The data mining task for discovering association rules consists of two steps: finding all frequent itemsets (i.e., all associations) and finding all rules whose confidence levels are at least a certain value, the *minimum confidence*. We use our group's ECLAT [11] association mining algorithm to compute the associations.

### 2.2 Similarity Metric

A measure of similarity between two entities reflects how close they are to one another. Let $X$ and $Y$ be two entities whose similarity we want to measure. We denote $Sim(X, Y)$ to mean the similarity measure between $X$ and $Y$. Ideally we would like $Sim$ to satisfy the following three properties:

- Identity: $Sim(X, Y) = 1$ corresponds to the fact that the two entities are identical in all respects.
- Distinction: $Sim(X, Y) = 0$ corresponds to the fact that the two entities are distinct in all respects.
- Relative Ordinality: If $Sim(X, Y) > Sim(X, Z)$, then it should imply that $X$ is more similar to $Y$ than it is to $Z$.

The first two properties bound the range of the measure while the third property ensures that similarities across objects can be meaningfully compared. This last property is particularly useful for clustering purposes.

Now we define our metric. Let $A$ and $B$ respectively be the association sets for a database $\mathcal{D}$ and that for a database $\mathcal{E}$. For an element $x \in A$ (respectively in $B$), let $\sup_{\mathcal{D}}(x)$ (respectively $\sup_{\mathcal{E}}(x)$) be the frequency of $x$ in $\mathcal{D}$ (respectively in $\mathcal{E}$). Define

$$Sim(A, B) = \frac{\sum_{x \in A \cap B} \max\{0, 1 - \alpha |\sup_{\mathcal{D}}(x) - \sup_{\mathcal{E}}(x)|\}}{\|A \cup B\|}$$

where $\alpha$ is a scaling parameter. The parameter $\alpha$ has the default value of 1 and is to reflect how significance the user view variations in supports are (the higher $\alpha$ is the more influential variations are). For $\alpha = 0$ the similarity measure is identical to $\frac{\|A \cap B\|}{\|A \cup B\|}$, i.e., support variance carries no significance.

### 2.3 Sampling and Association Rules

The use of sampling for approximate, quick computation of associations has been studied in the literature [10]. While computing the similarity measure, sampling can be used at two levels. First, if generating the associations is expensive (for large datasets) one can sample the dataset and subsequently generate the association set from the sample, resulting in huge I/O savings. Second, if the association sets are

large one can estimate the distance between them by sampling, appropriately modifying the similarity measure presented above. Sampling at this level is particularly useful in a distributed setting when the association sets, which have to be communicated to a common location, are very large.

## 3 Clustering Datasets

Clustering is commonly used for partitioning data [6]. The clustering technique we adopt is the simple tree clustering. We use the similarity metric of databases defined in Section 2 for as the distance metric for our clustering algorithm. Input to the algorithm is simply the number of clusters in the final result. At the start of the clustering process each database constitutes a unique cluster. Then we repeatedly merge the pair of clusters with the highest similarity and merge the pair into one cluster until there are the desired number of clusters left.

As our similarity metric is based on associations, there is an issue of how to merge their association lattices when two clusters are merged. A solution would be to combine all the datasets and recompute the associations, but this would be time-consuming and involve heavy communication overheads (all the datasets will have to be re-accessed). Another solution would be to intersect the two association lattices and use the intersection as the lattice for the new cluster, but this would be very inaccurate. We take the half-way point of these two extremes.

Suppose we are merging two clusters $\mathcal{D}$ and $\mathcal{E}$, whose association sets are respectively $A$ and $B$. The value of $\sup_{\mathcal{D}}(x)$ is known only for all $x \in A$ and that of $\sup_{\mathcal{E}}(x)$ is known only for all $x \in B$. The actual support of $x$ in the join of $\mathcal{D}$ and $\mathcal{E}$ is given as

$$\frac{\sup_{\mathcal{D}}(x) \cdot \|\mathcal{D}\| + \sup_{\mathcal{E}}(x) \cdot \|\mathcal{E}\|}{\|\mathcal{D}\| + \|\mathcal{E}\|}.$$

When $x$ does not belong to $A$ or $B$, we will approximate the unknown sup-value by a "guess" $\theta$ [2], which can be specific to the cluster as well as to the association $x$.

## 4 Experimental Analysis

In this section we experimentally evaluate our similarity metric[3]. We evaluate the performance and sensitivity of computing this metric using sampling in a distributed setting. We then apply our dataset clustering technique to synthetic datasets from IBM and on a real dataset from the Census Bureau, and evaluate the results obtained.

### 4.1 Setup

All the experiments (association generation, similarity computation) were performed on a single processor of a DECStation 4100 containing four 600MHz Alpha 21164 processors, with 256MB of memory per processor.

[2] We are evaluating two methods to estimate $\theta$. The strawman is to randomly guess a value between 0 and the minimum support. The second approach is to estimate the support of an item based on the available supports of its subsets.

[3] Due to lack of space we do not detail our experimentation on choice of $\alpha$.

We used different synthetic databases with size ranging from 3MB to 30MB, which are generated using the procedure described in [3]. These databases mimic the transactions in a retailing environment. Table 1 shows the databases used and their properties. The number of transactions is denoted as $numT$, the average transaction size as $T_l$, the average maximal potentially frequent itemset size as $I$, the number of maximal potentially frequent itemsets as $\|L\|$, and the number of items as Size. We refer the reader to [3] for more detail on the database generation.

| Database | $numT$ | $T_l$ | $I$ | $\|L\|$ | Size |
|---|---|---|---|---|---|
| D100 | 100000 | 8 | 2000 | 4 | 5MB |
| D200 | 200000 | 12 | 6000 | 2 | 12MB |
| D300 | 300000 | 10 | 4000 | 3 | 16MB |
| D400 | 400000 | 10 | 10000 | 6 | 25MB |

**Table 1.** Database properties

The Census data used in this work was derived from the County Business Patterns (State) database from the Census Bureau. Each dataset we derive (dataset per state) from this database contains one transaction per county. Each transaction contains items which highlight information on subnational economic data by industry. Each industry is divided into small, medium and large scale concerns. The original data has numeric data corresponding to number of such concerns occurring in the county. We discretize these numeric values into three categories: high, middle and low. So an item "high-small-agriculture" would correspond to a high number of small agricultural concerns. The resulting set of datasets have as many transactions as counties in the state and a high degree of associativity.

### 4.2 Sensitivity to Sampling Rate

In Section 2 we mentioned that sampling can be used at two levels to estimate the similarity efficiently in a distributed setting. If association generation proves to be expensive, one can sample the transactions to generate the associations and subsequently use these associations to estimate the similarity accurately. Alternatively, if the number of associations in the lattice are large, one can sample the associations to directly estimate the similarity. We evaluate the impact of using sampling to compute the approximate the similarity metric below.

For this experiment we breakdown the execution time of computing the similarity between two of our databases D300 and D400 under varying sampling rates. The two datasets were located in physically separate locations. We measured the total time to generate the associations for a minimum support of 0.05% (Computing Associations) for both datasets (run in parallel), the time to communicate the associations from one machine (Communication Overhead) to another and the time to compute the similarity metric (Computing Similarity) from these association sets. Transactional sampling influences the computing the associations while association sampling influences the latter two aspects of this experiment. Under association sampling, each processor computes a sample of its association set and sends it to the other, both then compute a part of similarity metric (in parallel). These two values are then merged appropriately, accounting for duplicates in the samples used. While both these sampling levels (transaction and association) could have different

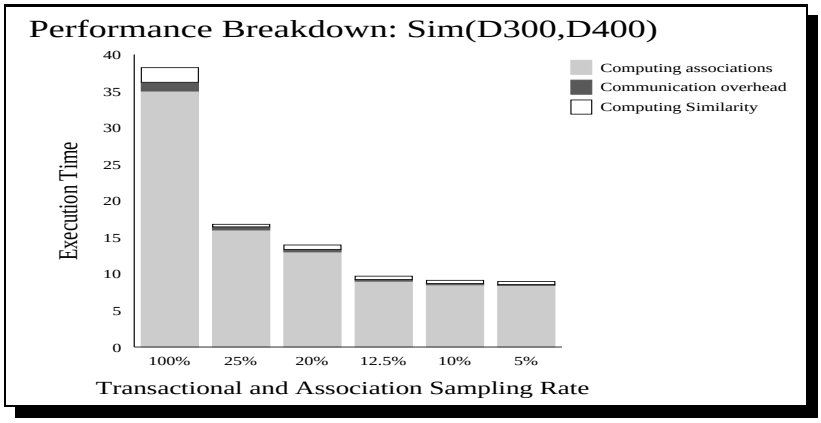



**Fig. 1.** Sampling Performance

sampling rates, for expository simplicity we chose to set both at a common value. We evaluate the performance under the following sampling rates, 5%, 10%, 12.5%, 20%, and 25%. Figure 1 shows the results from this experiment.

Breaking down the performance it is clear that by using sampling at both levels the performance improves dramatically. For a sampling rate of 10% the time to compute associations goes down by a factor of 4. The communication overhead goes down by a factor of 6 and the time to compute the similarity goes down by a factor of 7. This yields an overall speedup of close to 5. Clearly, the dominant factor in this experiment is computing the associations (85% of total execution time). However, with more traffic in the system, as will be the case when computing the similarity across several datasets (such as in clustering), and when one is modifying the probe set interactively, the communication overhead will play a more dominant role.

The above experiment affirms the performance gains from association and transactional sampling. Next, we evaluate the quality of the similarity metric estimated using such approximation techniques for two minimum support values (0.05% and 0.1%). From Table 2 it is clear that using sampling for estimating the similarity metric can be very accurate (within 2% of the ideal (**S**ampling **R**ate 100%)) for all sampling rates above 5%. We have observed similar results (speedup and accuracy) for the other dataset pairs as well.

| Support | SR-100% | SR-25% | SR-20% | SR-10% | SR-5% |
|---|---|---|---|---|---|
| 0.05% | 0.135 | 0.134 | 0.136 | 0.133 | 0.140 |
| 0.1% | 0.12 | 0.12 | 0.12 | 0.12 | 0.115 |

**Table 2.** Sampling Accuracy: Sim(D300,D400)

### 4.3 Synthetic Dataset Clustering

We evaluated the efficacy of clustering homogeneous distributed datasets based on similarity. We used the synthetic datasets described earlier as a start point. We randomly split each of the datasets D100, D200, D300, and D400 into 10 datasets of roughly equal size. For the sake of simplicity in exposition we describe only the

experiment that used only first three subsets from each. We ran a simple tree-based clustering algorithm on these twelve datasets. Figure 2 shows the result. The numbers attached to the joins are the *Sim* metric with $\alpha = 1.0$. Clearly the datasets from the same origin are merged first. Given four as the desired number of clusters (or a merge cutoff of 0.2), the algorithm stops right after executing all the merges depicted by full lines, combining all the children from the same parents into single clusters and leaving apart those from different parents. This experiment illustrates two key points. First, the similarity metric coupled with our merging technique seem to be an efficient yet effective way to cluster datasets. Second, hypothetically speaking, if these 12 datasets were representative of a distributed database, combining all 12 and mining for rules would have destroyed any potentially useful structural rules that could have been found if each cluster were mined independently (our approach).

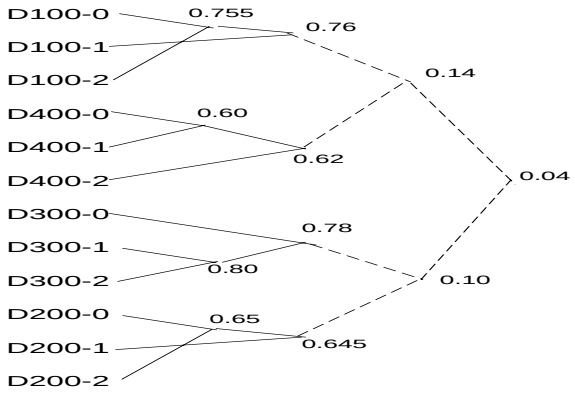



**Fig. 2.** Dataset Clustering

### 4.4 Census Dataset Evaluation

Table 3 shows the *Sim* values (with $\alpha = 1.0$) for a subset of the Census data for the year of 1988. As mentioned earlier each dataset corresponds to a state in the US. When asked to break the eight states into four clusters the clustering algorithm returned the clusters [IL, IA, TX], [NY, PA], [FL], and [OR,WA]. On looking at the actual *Sim* values it is clear that NY and PA have a closeted preference for one another IL, IA, and TX have strong preference for one another. OR has a stronger preference for IL, IA and TX, but once IL, IA, and TX were merged it preferred being merged with WA. Interestingly three pairs of neighboring states, i.e., (OR,WA), (IL,IA), and (NY,PA), are found in the same cluster.

An interesting by-play of discretization of the number of industrial concerns into three categories (high, middle and low) is that states with larger counties (area-wise), such as PA, NY and FL tend to have higher associativity (since each county has many items) and thereby tend to have less affinity to states with lower associativity. By probing the similarity between IA and IL further the most influential attribute is found to be agricultural concerns (no surprise there). The reason TX was found to be similar to these states was again due to agricultural concerns, a somewhat surprising result. However, this made sense, when we realized that cattle farming is also grouped

under agricultural concerns! Interestingly, we found that the Census data benefited, performance-wise, from association sampling due its high associativity.

| State | IL | NY | PA | FL | TX | OR | WA |
|---|---|---|---|---|---|---|---|
| IA | 0.54 | 0.01 | 0.01 | 0.16 | 0.44 | 0.26 | 0.1 |
| IL | | 0.02 | 0.02 | 0.24 | 0.52 | 0.30 | 0.16 |
| NY | | | 0.31 | 0.14 | 0.01 | 0.04 | 0.08 |
| PA | | | | 0.05 | 0.01 | 0.03 | 0.04 |
| FL | | | | | 0.24 | 0.21 | 0.21 |
| TX | | | | | | 0.32 | 0.16 |
| OR | | | | | | | 0.25 |

**Table 3.** Census Dataset: *Sim* Values (support = 20%)

## 5 Conclusions

In this paper we propose a method to measure the similarity among homogeneous databases and show how one can use this measure to cluster similar datasets to perform meaningful distributed data mining. An interesting feature of our algorithm is the ability to interact via informative querying to identify attributes influencing similarity. Experimental results show that our algorithm can adapt to time constraints by providing quick (speedup of 5-7) and accurate estimates (within 2%) of similarity. We evaluate our work on several datasets, synthetic and real, and show the effectiveness of our techniques. As part of future work we will focus on evaluating and applying dataset clustering to other real world distributed data mining tasks. It seems likely that the notion of similarity introduced here would work well for tasks such as Discretization and Sequence Mining with minor modifications if any. We are also evaluating the effectiveness of the merging criteria described in Section 3.

## References


1. C. Aggarwal and P. Yu. Online generation of association rules. In *ICDE*'98.
2. R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms*, 1993.
3. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In U. Fayyad and et al, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press, Menlo Park, CA, 1996.
4. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th VLDB Conf.*, 1994.
5. G. Das, H. Mannila, and P. Ronkainen. Similarity of attributes by external probes. In *KDD* 1998.
6. U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. The KDD process of rextracing useful information from volumes of data. *Communications of ACM*, 39(11):27–34, 1996.
7. R. Goldman, N.Shivakumar, V. Suresh, and H. Garcia-Molina. Proximity search in databases. In *VLDB Conf.*, 1998.
8. H. Jagadish, A. Mendelzon, and T. Milo. Similarity based queries. In *PODS*, 1995.
9. R. Subramonian. Defining *diff* as a data mining primitive. In *KDD* 1998.
10. H. Toivonen. Sampling large databases for association rules. In *VLDB Conf.*, 1996.
11. M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD*, 1997.

# Scalable Model for Extensional and Intensional Descriptions of Unclassified Data

Hércules A. Prado[1,2,*], Stephen C. Hirtle[2,**], Paulo M. Engel[1,***]

[1] Universidade Federal do Rio Grande do Sul
Instituto de Informática
Av. Bento Gonçalves, 9500 - Bairro Agronomia
Porto Alegre / RS - Brasil
Caixa Postal 15.064 - CEP 91.501-970
Fone: +55(051)316-6829
Fax: +55(051)319-1576
[2] University of Pittsburgh
Department of Information Sciences and Telecommunications
135 North Bellefield Ave. Pittsburgh, PA 15.260
Phone: +1(412)624-9434
Fax: +1(412)624-2788
{prado, engel}@inf.ufrgs.br and hirtle+@pitt.edu

**Abstract.** Knowledge discovery from unlabeled data comprises two main tasks: identification of "natural groups" and analysis of these groups in order to interpret their meaning. These tasks are accomplished by unsupervised and supervised learning, respectively, and correspond to the taxonomy and explanation phases of the discovery process described by Langley [9]. The efforts of Knowledge Discovery from Databases (KDD) research field has addressed these two processes into two main dimensions: (1) scaling up the learning algorithms to very large databases, and (2) improving the efficiency of the knowledge discovery process. In this paper we argue that the advances achieved in scaling up supervised and unsupervised learning algorithms allow us to combine these two processes in just one model, providing extensional (who belongs to each group) and intensional (what features best describe each group) descriptions of unlabeled data. To explore this idea we present an artificial neural network (ANN) architecture, using as building blocks two well-know models: the ART1 network, from the Adaptive Resonance Theory family of ANNs [4], and the Combinatorial Neural Model (CNM), proposed by Machado ([11] and [12])). Both models satisfy one important desiderata for data mining, learning in just one pass of the database. Moreover, CNM, the intensional part of the architecture, allows one to obtain rules directly from its structure. These rules represent the insights on the groups. The architecture can be extended to other supervised/unsupervised learning algorithms that comply with the same desiderata.

* Researcher at EMBRAPA — Brazilian Enterprise for Agricultural Research and lecturer at Catholic University of Brasília (Supported by CAPES - Coordenaçao de Aperfeiçoamento de Pessoal de Nível Superior, grant nr. BEX1041/98-3)
** Professor at University of Pittsburgh
*** Professor at Federal University of Rio Grande do Sul

## 1 Introduction

Research in Knowledge Discovery from Databases (KDD) has developed along two main dimensions: (1) improving the knowledge discovery process, and (2) scaling up this same process to very large databases. Machine Learning, as an important field related to KDD, is founded on three principles [19]:

1. Modeling of cognitive processes, aiming to select characteristics of interest to be formalized as knowledge;
2. Computer science, which offers a formalism to support the descriptions of those characteristics, as well as providing approaches to evaluate the degree of computational difficulty of the issues involved; and
3. Applications, where one departs from practical needs to the implementation of systems.

In this article, we depart from a characterization of the concept formation activity as a cognitive process, proposing a computational approach to support this activity, from the point of view of KDD. We take the concept of performance as given by the relation functionality/resources applied. By this way, we present a model where funcionality is increased while the resources applied are just slightly changed.

## 2 Motivation

According to Wrobel [19], a concept is ”a generalized description of sets of objects”. In this sense, Easterlin and Langley [5] analyse the concept formation process as follows:

1. Given a set of objects (instances, events, cases) descriptions, usually presented incrementally;
2. find sets of objects that can be grouped together (aggregation), and
3. find intensional description of these sets of objects (characterization).

Murphy and Medin [14] discuss two hypothesis that constrain the way objects are grouped in concepts:

1. Similarity hypothesis: this hypothesis sustain that what defines a class is that its members are similar to each other and not similar to members of other classes.
2. Correlated attribute hypothesis: this hypothesis states that ”natural groups” are described according to clusters of features and that categories reflect the cluster structure of correlations.

The first hypothesis presents a problem that is: the similarity criteria must be applied to a pre-defined set of features and the definition of this set is affected by the previous knowledge one has over the objects. However, when just a small knowledge about the data exists, this criteria is used as a first approximation. Over this approximation, the correlated attribute hypothesis is applied. In a broad sense, what is desirable in this process is to provoke the mental operations that can lead to a problem solution ([16] and [15]). Actually, since it seems that the discovery process, as a rule, requires the human judgment [9], it is useful to leave available to the analyst all relevant information to evaluate both hypothesis when searching for the classes' structures.

## 3 Proposed Architecture

The research on the KDD realm has emphasized improvements in the processes of supervised and unsupervised learning. More recently, many unsupervised learning algorithms have been scaled up according to the desiderata proposed by Agrawal *et al.* [1] for this kind of learning algorithm. Considering these advances and the ones in supervised learning, we believe there is enough room to scale up the combined process of unsupervised and supervised learning in order to obtain better descriptions of unclassified data. By ”better descriptions” we mean obtaining intensional (what are the main characteristics of each class) descriptions, beyond the extensional (what objects are members of each class) ones, usually provided. We explore our idea with a hybrid architecture, based into two well-know models: ART1 [4], used for cluster binary data, and CNM ([11] and [12]), used to map the input space in the formed classes. Both ANNs present an important characteristic to support data mining: they learn in just one pass of the entire data set. The model is illustrated by Figure 1.

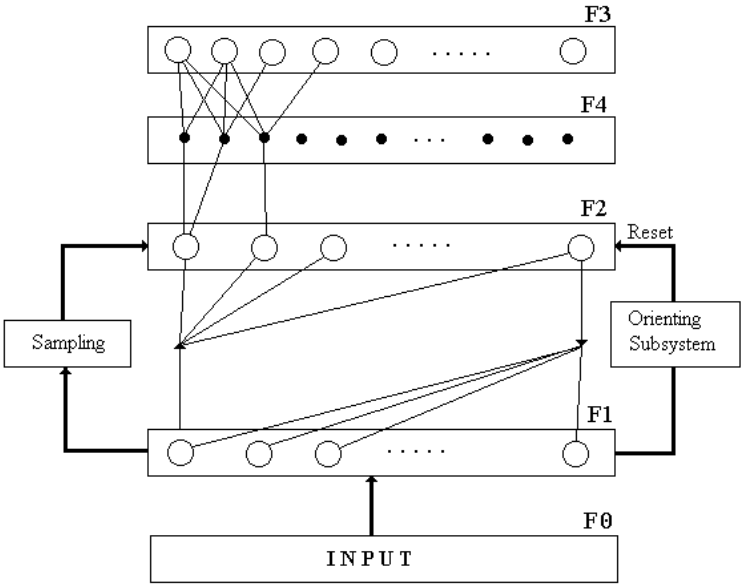



**Fig. 1.** Describing unclassified data

The architecture is composed by five layers according to the schema: Input layer (F0, where the examples are introduced in the architecure); Aggregation module (F1 and F2, where the classes are defined); Characterization module (F2, F3 , and F4, where the classes are explained).

For the characterization module, it requires the pre-existence of classes that would not be available when the process starts. We overcome this problem by creating the

classes by means of a sampling subsystem. The creation of classes through sampling was explored by Guha [8] with consistent results. As a consequence of the sampling process, a complete execution of the system will take more than one pass of the input data set. Considering $\gamma$ the size of the sample and **D** the size of the input data set, a complete execution of the system will take, precisely, $\mathbf{D} + \gamma$ records. In the next two sections, we describe each model used as building blocks for our architecture.

## 4 ART1 Neural Network

ART1 (F1 and F2 layers) is a member of the so-called ART family, that stands by Adaptive Resonance Theory [10], [3], [4] and [6].

ART1 is a competitive recurrent network with two layers, the input layer and the clustering layer. This network was developed to overcome the plasticity-stability dilemma [7], allowing an incremental learning, with a continuous updating of the clusters prototypes, and preserving the previously stored patterns. The clustering algorithm proceeds, in general steps, as folows: (a) the first input is selected to be the first cluster; (b) each next input is compared with each existing cluster; the first cluster where the distance to the input is less then a threshold is chosen to cluster the input. Otherwise, the input defines a new cluster. It can be observed that the number of clusters depends on the threshold and the distance metric used to compare the inputs with the clusters. For each input pattern presented to the network, one output unit is declared winner (at the first pattern, the own input pattern defines the cluster). The winner backpropagates a signal that encodes the expected pattern template. If the current input pattern differs more than a defined threshold from the backpropagated signal, the winner are temporarily disabled (by the Orienting System) and the next closest unit is declared winner. The process continues until an output unit become a winner, considering the threshold. If no one of the output units become a winner, a new output unit is defined to cluster the input pattern. Graphically, an ART1 network can be illustrated by Figure 2, where it appears with four input and six ouput neurons. $t_{ij}$ and $b_{ij}$ are, respectively, bottom-up and top-down connections.

One important characteristic of this ANN is that it works in just one pass, what is interesting when we are processing a huge amount of data. The training algorithm for this network is the following:

- **Step1. Initialization**: The bottom-up $b_{ij}(t)$ and top-down $t_{ij}(t)$ weight connection between input node $i$ and output node $j$ at time $t$ are set up. The fraction $\rho$ (vigilance parameter) is defined, indicating how close an input must be to a stored exemplar to match. Initialize $N$ and $M$, numbers of input and output nodes.
- **Step 2. Apply New Input**
- **Step 3. Compute Matching Scores**: The bottom-up weights are applied to the input pattern, generating the output signal: $\mu_j$.
- **Step 4. Select Best Matching Exemplar**: $\mu_j^* = \max\{\mu_j\}$ is taken as the best exemplar.
- **Step 5. Vigilance Test**: The best exemplar and the input pattern are compared, according to $\rho$. If the distance is acceptable, the control flows to **Step 7**, otherwise **Step 6** proceeds.

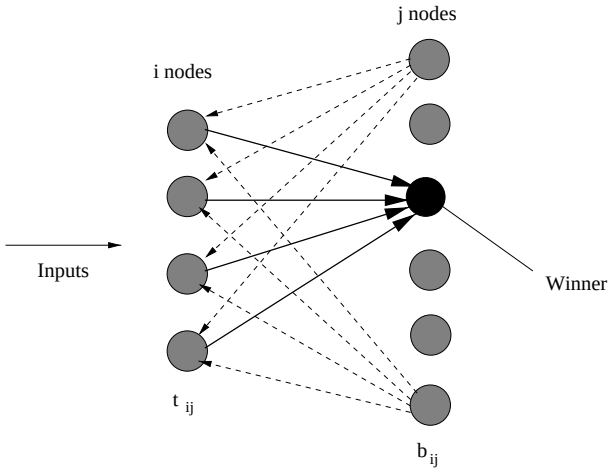



**Fig. 2.** Architecture of an ART network [3]

- **Step 6. Disable Best Matching Exemplar**: The ouput of the best matching node selected in Step4 is temporarily set to zero and no longer takes part in the maximization of Step4. Then go to Step 3.
- **Step 7. Adapt Best Matching Exemplar**:

$$t_{ij_*}(t+1) = t_{ij_*}(t)t(x_i)$$
$$b_{ij_*}(t+1) = \frac{t_{ij_*}(t)x_i}{\frac{1}{2} + \sum_{i=0}^{N-1} t_{ij_*}(t)x_i}$$

- **Step 8. Repeat by Going to Step 2**: First enable any node disabled in Step 6.

## 5 Combinatorial Neural Model (CNM)

CNM (F2, F3, and F4 layers) is a hybrid architecture for intelligent systems that integrates symbolic and connectionist computational paradigms. This model is able to recognize regularities from high-dimensional symbolic data, performing mappings from this input space to a lower dimensional output space. Like ART1, this ANN also overcomes the plasticity-stability dilemma [7].

The CNM uses supervised learning and a feedforward topology with: one input layer, one hidden layer - here called combinatorial - and one output layer (Figure 3). Each neuron in the input layer corresponds to a concept - a complete idea about an object of the domain, expressed in an object-attribute-value form. They represent the evidences of the domain application. On the combinatorial layer there are aggregative fuzzy AND neurons, each one connected to one or more neurons of the input layer by arcs with adjustable weights. The output layer contains one aggregative fuzzy OR neuron for each possible class (also called hypothesis), linked to one or more neurons on the combinatorial layer. The synapses may be excitatory or inhibitory and they are

characterized by a strength value (weight) between zero (not connected) to one (fully connected synapses).

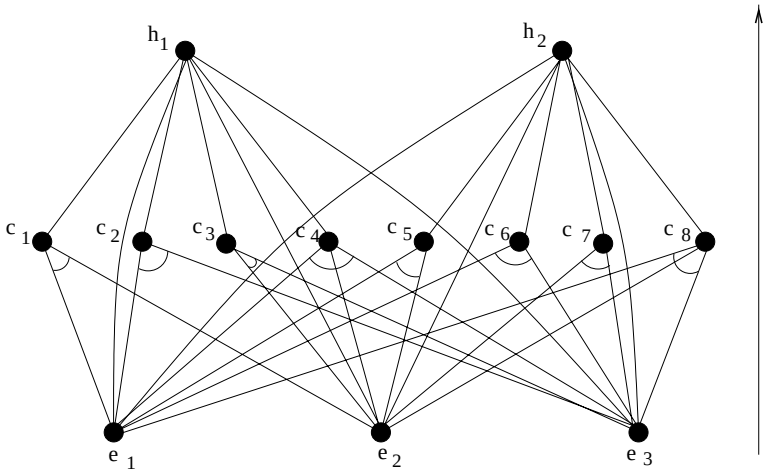



**Fig. 3.** Complete version of CNM for 3 input evidences and 2 hypotheses [11]

The network is created completely uncommited, according to the following steps: (a) one neuron in the input layer for each evidence in the training set; (b) a neuron in the output layer for each class in the training set; and (c) for each neuron in the output layer, there is a complete set of hidden neurons in the combinatorial layer which corresponds to all possible combinations (length between two and nine) of connections with the input layer. There is no neuron in the combinatorial layer for single connections. In this case, input neurons are connected directly to the hypotheses.

The learning mechanism works in only one iteration, and it is described bellow:

**PUNISHMENT_AND_REWARD_LEARNING_RULE**

– **Set** to each arc of the network an accumulator with initial value zero;
– **For each** example case from the training data base, **do**:
  - *Propagate* the evidence beliefs from input nodes until the hypotheses layer;
  - **For each** arc reaching a hypothesis node, **do**:
    * **If** the reached hypothesis node corresponds to the correct class of the case
    * **Then** *backpropagate* from this node until input nodes, increasing the accumulator of each traversed arc by its evidencial flow (Reward)
    * **Else** *backpropagate* from the hypothesis node until input nodes, decreasing the accumulator of each traversed arc by its evidencial flow (Punishment).

After training, the value of accumulators associated to each arc arriving to the output layer will be between [-T, T], where T is the number of cases present in the training set. The last step is the prunning of network; it is performed by the following actions: (a) remove all arcs whose accumulator is lower than a threshold (specified by a specialist); (b) remove all neurons from the input and combinatorial layers that became disconnected from all hypotheses in the output layer; and (c) make weights of the arcs

arriving at the output layer equal to the value obtained by dividing the arc accumulators by the largest arc accumulator value in the network. After this pruning, the network becomes operational for classification tasks. This ANN has been applied with success in data mining tasks ([2], [17], and [18]).

## 6 Ongoing Work

This paper presents an architecture to scale up the whole process of concept formation according two main constraints: the identification of groups composed by similar objects and the description of this groups by the higher correlated features. The ongoing work includes the implementation and evaluation of this architecture, instantiated to ART1 and CNM, and its extension to cope with continuous data.

## References


1. Agrawal, R., Gehrke, J., Gunopulos, D., Raghanavan, P. Automatic Subspace Clustering of High-Dimensional Data for Data Mining Applicati ons. In: Proceedings of ACM SIGMOD98 International Conference on Management of Data, Seattle, Washington, 1998.
2. Beckenkamp, F. G., Feldens, M. A., Pree, W.: Optimizations of the Combinatorial Neural Model. IN: Vth Brazilian Symposium on Neural Networks. (SBRN' 98), Belo Horizonte, Brazil.
3. Bigus, J. P. Data Mining with Neural Networks. [S.l.]: McGraw-Hill, 1996. p.3-42.
4. Carpenter, G. and Grossberg, S. Neural Dynamics of Category Learning and Recognition: Attention, Memory, Consolidation, and Amnesia. In: Joel L. Davis (ed.), Brain structure, learning, and memory. AAAS Symposia Series, Boulder, CO: Westview Press, 1988. p.233-287.
5. Easterlin, J.D., Langley, P.: A Framework for Concept Formation. In: Seventh Annual Conference of the Cognitive Science Society, Irvine, CA, 1985.
6. Engel, P. M. Lecture Notes. Universidade Federal do Rio Grande do Sul. Porto Alegre-RS, Brazil: CPGCC da UFRGS, 1997.
7. Freeman, J. A., Skapura, D. M.: Neural Networks, Algorithms, Applications, and Program Techniques. [S.l.]: Addison-Wesley, 1992. p.292-339.
8. Guha, S., Rastogi, R., Shim, K. Cure: An Efficient Clustering Algorithm for Larg e Databases. In: Proceedings of ACM SIGMOD98 International Conference on Managemen t of Data, Seattle, Washington, 1998.
9. Langley, P. The Computer-Aided Discovery of Scientific Knowledge. In: Proc. of the First International Conference on Discovery Science, Fukuoka, Japan, 1998.
10. Lippmann, D. An Introduction to Computing with Neural Nets, IEEE ASSP Magazine. April, 1987.
11. Machado, R. J., Rocha, A. F.: Handling knowledge in high order neural networks: the combinatorial neural network. Rio de Janeiro: IBM Rio Scientific Center, Brazil, 1989. (Technical Report CCR076).
12. Machado, R. J., Carneiro, W., Neves, P. A.: Learning in the combinatorial neural model, IEEE Transactions on Neural Networks, v.9, p.831-847. Sep.1998.
13. Medin, D., Altom, M.W., Edelson, S.M. and Freko, D. Correlated symptoms and simulated medical classification. Journal of Experimental Psychology: Learning, Memory and Cognition, 8:37-50, 1983.
14. Murphy, G. and Medin, D.: The Role of Theories in Conceptual Coherence. Psychological Review, 92(3):289-316, July, 1985.

15. Pereira, W. C. de A. Resoluçao de Problemas Criativos: Ativaçao da Capacidade de Pensar. Departamento de Informaçao e Documentaçao/EMBRAPA, Brasília-DF, 1980. 54pp.
16. Polya, G. How to Solve It: A New Aspect of Mathematical Method. Princeton: Princeton University Press, 1972. 253pp.
17. Prado, H. A., Frigeri, S. R., Engel, P. M.: A Parsimonious Generation of Combinatorial Neural Model. IN: IV Congreso Argentino de Ciencias de la Computación (CACIC' 98), Neuquén, Argentina, 1998.
18. Prado, H. A. do; Machado, K.F.; Frigeri, S. R.; Engel, P. M. Accuracy Tuning in Combinatorial Neural Model. PAKDD' 99 - Pacific-Asia Conference on Knowledge Discovery and Data Mining. Proceedings ... Beijing, China, 1999
19. Wrobel, S. Concept Formation and Knowledge Revision. Dordrecht, The Netherlands: Kluwer, 1994. 240pp.

# Parallel Data Mining of Bayesian Networks from Telecommunications Network Data

Roy Sterritt, Kenny Adamson, C. Mary Shapcott, and Edwin P. Curran

University of Ulster,
Faculty of Informatics, Newtownabbey, County Antrim, BT37 0QB, Northern Ireland.
{r.sterritt, k.adamson, cm.shapcott, ep.curran}@ulst.ac.uk
http://www.ulst.ac.uk/

**Abstract.** Global telecommunication systems are built with extensive redundancy and complex management systems to ensure robustness. Fault identification and management of this complexity is an open research issue with which data mining can greatly assist.
This paper proposes a hybrid data mining architecture and a parallel genetic algorithm (PGA) applied to the mining of Bayesian Belief Networks (BBN) from Telecommunication Management Network (TMN) data.

## 1 Introduction and the Global Picture

High-speed broadband telecommunication systems are built with extensive redundancy and complex management systems to ensure robustness. The presence of a fault may not only be detected by the offending component and its parent but the consequence of that fault discovered by other components. This can potentially result in a net effect of a large number of alarm events being raised and cascaded to the element controller, possibility flooding it in a testing environment with raw alarm events. In an operational network a flood is prevent by filtering and masking functions on the actual network elements (NE). Yet there can still be a considerable amount of alarms depending on size and configuration of the network, for instance, although unusual to execute, there does exist the facility for the user to disable the filtering/masking on some of the alarm types.

The behaviour of the alarms is so complex it appears non-deterministic. It is very difficult to isolate the true cause of the fault/multiple faults. Data mining aims at the discovery of interesting regularities or exceptions from vast amounts of data and as such can assist greatly in this area.

Failures in the network are unavoidable but quick detection and identification of the fault is essential to ensure robustness. To this end the ability to correlate alarm events becomes very important.

This paper will describe how the authors, in collaboration with NITEC, a Nortel Networks R&D lab, have used parallel techniques for mining bayesian networks from telecommunications network data. The primary purpose being fault management - the induction of a bayesian network by correlating offline

event data, and deducing the cause (fault identification) using this bayesian network with live events.

The problems encountered using traditional computing and how these can be overcome using a high performance computing architecture and algorithm are reported along with the results of the architecture and parallel algorithm.

### 1.1 Telecommunication Fault Management, Fault Correlation and Data Mining BBNs

Artificial intelligence and database techniques are useful tools in the interrogation of databases for hitherto unseen information to support managerial decision making or aid advanced system modelling. Knowledge discovery in databases is a technique for combing through large amounts of information for relationships that may be of interest to a domain expert but have either been obscured by the sheer volume of data involved or are a product of the volume[1].

Bayesian Belief Networks is a technique for representing and reasoning with uncertainty[2]. It represents CAUSEs and EFFECTs as nodes and connects CAUSEs and EFFECTs as networks with a probability distribution. Bayesian Belief Networks have been successfully used to build applications, such as medical diagnosis, where multiple causes bear on a single effect[2]. We proposed to employ both techniques in conjunction to model complex systems that act in a non-deterministic manner. These problems are common to real-life industrial systems that produce large amounts of information. The data-handling requirements, computationally expensive techniques and real-time responsiveness make parallel processing a necessity.

The exemplar under which the architecture is being developed is a telecommunications application, involving the Synchronous Digital Hierarchy (SDH) - the backbone of global communications. It offers flexibility in dealing with existing bandwidth requirements and provide capabilities for increasingly sophisticated telecommunications services of the future[3]. One key area of interest to engineers is the management of events and faults in a network of SDH multiplexers[4]. An event is a change of status within a network that produces a corresponding alarm message or fault. When a network error occurs, each multiplexer determines the nature of the problem and takes steps to minimise any loss in signal or service. To facilitate the process of error recovery, there are many levels of redundancy built into a network, e.g. signal re-routing and the use of self-healing rings. The management of faults is complex because :

- the occurrence of faults is time-variant and non-deterministic;
- faults can produce a cascade of other faults;
- fault-handling must be performed in real-time.

Although the behaviour of individual multiplexers is accurately specified, the complex interactions between a number of multiplexers and the variability of real-life network topologies means that the process of fault occurrence is more complicated than a single specification will imply. This problem is compounded

by the growth of networks and the increasing variety of network topology. The application of data mining and in particular parallel data mining can assist greatly.

Data mining aims at the discovery of interesting regularities or exceptions from vast amounts of data. As has been stated, fault management is a critical but difficult area of telecommunication network management since networks produce a vast quantity of data that must be analysed and interpreted before faults can be located. Alarm correlation is a central technique in fault identification yet it is difficult to cope with incomplete data and the sheer complexity involved.

At the heart of alarm event correlation is the determination of the cause. The alarms represent the symptoms and as such are not of general interest[5]. There are two real world concerns, the;

- sheer volume of alarm event traffic when a fault occurs;
- cause not the symptoms.

A technique that can tackle both these concerns would be best, yet this can be difficult to achieve.

### 1.2 The Architecture

We proposed a parallel mining architecture for the elucidation of an accurate system representation based primarily on Bayesian Belief Networks that are induced using Knowledge Discovery techniques. The architecture has a modular design that can be reconfigured according to application specification. The system was prototyped using the INMOS transputer as the target hardware. Within the telecommunications domain, it is hoped that the application of the system will ultimately assist in fault management but also in the analysis of test data.

The actual realisation of the architecture for the situation is shown in Figure 1. It can be seen that the input data is available in the form of log data from the element controller. The team identified a need for the efficient preparatory processing of data that appeared in the event log format. This led to the design and preliminary implementation of a data cleaner and a data pre-processor. The data cleaner allows a user to specify the format of a text document in a generic way using a template, and to specify filtering conditions on the output. In the telecommunications application the text document is an event log, and the template file can be altered if the structure of the event records changes. The data cleaner parses a log file and passes the resulting events to the pre-processor which time-slices the information and creates an intermediate file for use in the induction module.

The Bayesian net, created as a result of induction, is potentially useful in a fault situation where the faults most likely to be responsible for observed alarms can be computed from the net and relayed to a human operator. For this reason there is a deduction module in the architecture whereby observed conditions in the telecommunications network can be fed into the Bayesian net and changes in the probabilities of underlying fault conditions can be computed[6].

However, the components are able to operate in isolation, provided that they are provided with files in the correct input format. In particular the induction component and the deduction component use data in Bayesian Network Interchange format[7].

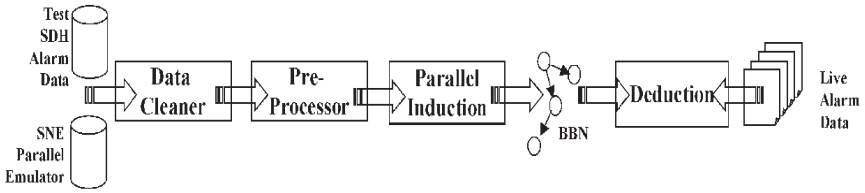



**Fig. 1.** The Architecture

## 2 The Parallel Data Mining Algorithm

### 2.1 The Need for Parallelism

In this case, as in many others, the structure of the graphical model (the Bayesian net) is not known in advance, but there is a database of information concerning the frequencies of occurrence of combinations of different variable values (the alarms). In such a case the problem is that of induction - to induce the structure from the data. Heckerman has a good description of the problem[8]. There has been a lot of work in the literature in the area, including that of Cooper and Herskovits[9]. Unfortunately the general problem is NP-hard [10]. For a given number of variables there is a very large number of potential graphical structures which can be induced. To determine the best structure then in theory one should fit the data to each possible graphical structure, score the structure, and then select the structure with the best score. Consequently algorithms for learning networks from data are usually heuristic, once the number of variables gets to be of reasonable size. There are $2^{k(k-1)/2}$ distinct possible independence graphs for a k-dimensional random vector: this translates to 64 probabilistic models for k= 4, and 32, 768 models for k = 6. Several different algorithms were prototyped and tested using the telecommunications data but since the potential number of graph candidates is so large a genetic algorithm was developed.

### 2.2 Parallel Cause And Effect Genetic Algorithm (P-CAEGA)

Goldberg describes many ways to view genetic algorithms (GA) [11]: as problems solvers, as a basis for competent machine learning, as a computational model of innovation and creativity and so on. In the work described here the problem is to find the best cause-and-effect network in a very large solution space of

all possible cause-and-effect networks since the problem is NP-hard a heuristic search technique must be used. This led to a consideration of genetic algorithms, since they have been shown to work well in many application areas[12] and offers a robust means of searching for the globally optimal solution[13].

The genetic algorithm works on a population of solutions, which change as the algorithm cycles through a sequence of generations, until a satisfactory solution has been found. Initialisation consists of the creation of an initial population, a pool of breeders. In each generation each breeder is scored, and the best breeders are selected (possibly with some uncertainty) to breed and create solutions for the next generation. A solution created by breeding is a genetic mixture of its two parents, and may also have been subject to random mutations which allow new gene sequences to be created. Solutions are directed graphs, viable solutions (those which will be scored and allowed to breed in the next generation) are directed acyclic graphs. The scoring function used was an adaptation of one proposed by Cooper and Herskovits[9], in which the best fit to the experimental data is calculated using Bayesian techniques. High scoring structures have a greater chance of being selected as parents for the next generation.

Due to the sheer volume of data involved in data mining[1], the time required to execute genetic algorithms and the intrinsic parallel nature of genetic algorithms[14], it was decided to implement a parallel version of the CAEGA algorithm (P-CAEGA).

There are a number of approaches to parallelising an algorithm for execution on more than one processor. An architecture with common memory can be used which allows efficient communication and synchronisation. Unfortunately these systems cannot be scaled to a large number of processors because of physical construction difficulties and contention for the memory bus[15].

An alternative is the distributed programming model or message passing model. Two environments widely researched in this area are the Local Area Network (LAN) using Parallel Virtual Machine (PVM) and Transputer Networks using Inmos development languages[16]. In the network architecture the hardware scales easily to a relatively large number of processors but this is eventually limited because of network contention. The Transputer hardware is a point-to-point architecture with dedicated high-speed communications with no contention and no need for addressing. The system can be highly scaleable if the program is constructed accordingly.

The parallel prototype implementation was carried out on T805 INMOS Transputers connected to a Sun Workstation with development performed in parallel C. The sequential prototype of CAEGA had been coded in Pascal. This was converted to C. The C code was then decomposed into processes that needed to be run sequentially and those that could be executed in parallel. Communications channels were used to transfer data between processes.

The first parallel version is a straightforward master-slave (processor farm) implementation. Breeding (reproduction, crossover and mutation) was carried out in parallel. In fact the scoring was also implemented in parallel. The selection had to be implemented sequentially and thus remained on the master

(the root processor which is the controller, and is connected to the host). This was necessary, as all of the structures from the new generation needed to be remixed to form new parents from the gene pool before distribution to the slaves for breeding. The remaining processors are utilised as slaves, which carry out the breeding in parallel and report the new structures and their scores to the master (root processor).

As was anticipated from preliminary investigations the scaleability achievable is limited because of the overhead of communications. For less than 8 processors the echo n-2 holds (excluding the master; n-1). It is believed, with further work on the efficiency of the algorithm this could be improved, but a linear increase (excluding the master) is not expected because of the sheer amount of communications involved.

This implementation represents a straight forward first prototype. It is a direct parallelisation of an CAEGA which did not change the underlying nature of the algorithm. This has resulted in global communications, which limits the scaleable - speedup ratio. In a LAN implementation this would be even more restrictive due to communications costs. In general an effective concurrent design will require returning to first principles.

### 2.3 Results

Typical results of the application of the algorithms described are shown below. The data which is shown results from an overnight run of automated testing, but does not show dependencies on underlying faults. About 12,000 individual events were recorded and the graph in Figure 2 shows a generated directed graph in which the width of the edge between two nodes (alarms) is proportional to the strength of connection of the two variables. It can be seen that PPI-AIS and LP-EXC have the strongest relationship, followed by the relationship between PPI-Unexp-Signal and LP-PLM. Note that the directions of the arrows are not important as causal indicators but variables sharing the same parents do form a group.

The graph in Figure 3 shows the edge strengths as strongest if the edge remains in models which become progressively less complex - where there is a penalty for complexity. It can be seen that the broad patterns are the same in the two graphs but that the less strong edges are different in the two graphs. In the second graph the node NE-Unexpected-Card shows links to three other nodes, whereas it has no direct links in the first graph.

The results from an industrial point of view are very encouraging. The case for using a genetic algorithm holds and parallelising it speeds up this process. The algorithm is currently being used by NITEC to analyse their data produced by a SDH network when a fault occurs. From their point of view it has been a worthwhile effort for the speed-up.

It has been established that genetic scoring and scaling could be implemented in parallel but the communications cost in transmitting these back to the master removed any benefit from just having the master perform these functions.

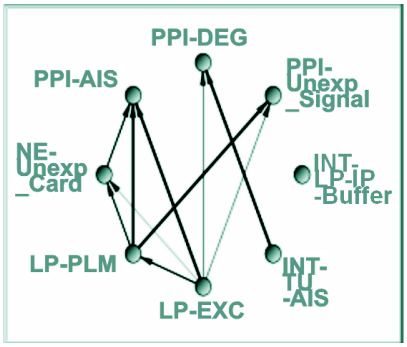



**Fig. 2.** Example results BBN of TMN data

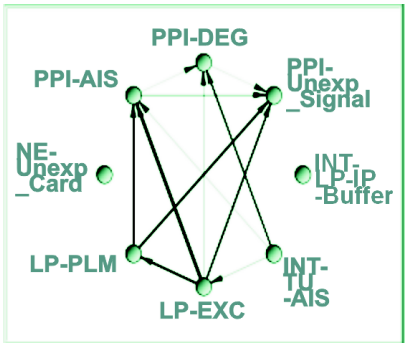



**Fig. 3.** Another set of results

### 2.4 Future Potential Research

This study assessed GAs as a solution provider. As Goldberg states "some of us leave the encounter with far more"[11] . Future development will take the basic solution further and remove the limitation on scaleability. What is required is a concurrent algorithm as opposed to a modification of a sequential algorithm. The next planned development is a redesign of the algorithm into the "continental" algorithm (current development name).

P-CAEGA as it stands can be classified as global parallelisation. Every individual has a chance to mate with all the rest (i.e. random breeding), thus the implementation did not affect the behaviour of the original algorithm (CAEGA).

The "continental" version would be a more sophisticated parallel approach where the population is divided into sub-populations , relatively isolated from each other. This model introduces a migration element that would be used to send some individuals from one sub-population to another. This adapted algorithm would yield local parallelism and each processor could be thought of as a continent where the majority of the breeding occurs between residents with limited migration. This modification to the behaviour of the original algorithm would vastly decrease the communications cost, and present a more scaleable implementation.

## 3 Conclusion

The association between the University of Ulster and Nortel in the area of fault analysis has continued with the Garnet project. Garnet is using the techniques developed in NETEXTRACT to develop useful tools in the area of live testing. The basic idea is to regard the Bayesian nets as abstract views of the test network's response to stimuli. In a further development, the Jigsaw project entails

the construction of a data warehouse for the storage of the test data, with a direct link to data mining algorithms.

Although this paper has described the work completed with reference to Nortel's telecommunications network, the architecture is generic in that it can extract cause-and-effect nets from large, noisy databases, the data arising in many areas of science and technology, industry and business and in social and medical domains. The corresponding hypothesis to the aim of this project could be proposed - that cause and effect graphs can be derived to simulate domain experts' knowledge and even extend it.

The authors would like to thanks Nortel Networks and the EPSRC for their support during the project.

## References

1. Agrawal, R., Imielinski, T., Swami, A.,"Database Mining: a Performance Perspective",IEEE Trans. KDE, 5(6), Dec, pp914-925, 1993.
2. Guan, J.W., Bell, D.A., Evidence Theory and Its Applications, Vol. 1&2, Studies in Computer Science and AI 7,8, Elsevier, 1992.
3. ITU-T Rec. G.782 Types and General Characteristics of SDH Muliplexing Equipment, 1990.
4. ITU-T Rec. G.784 SDH Management, 1990.
5. Harrison K. "A Novel Approach to Event Correlation", HP Labs., Bristol. HP-94-68, July, 1994, pp. 1-10.
6. McClean, S.I. and Shapcott, C.M., 1997, "Developing BBNs for Telecommunications Alarms", Proc. Conf. Causal Models & Statistical Learning, pp123-128, UNICOM, London.
7. MS Decision Theory and Adaptive Systems Group, 1996. "Proposal for a BBN Interchange Format", MS Document.
8. Heckerman D, 1996. "BBNs for Knowledge Discovery" In Fayyad UM, Piatetsky-Shapiro G, Smyth P and Uthurusamy R (Eds.), Advances in Knowledge Discovery and Data Mining AAAI Press/MIT Press, pp273-305.
9. Cooper, G.F. and Herskovits, E., 1992. "A Bayesian Method for the Induction of Probabilistic Networks from Data". ML, 9, pp309-347
10. Chickering D.M. and D. Heckerman, 1994. "Learning Bayesian networks is NP-hard". MSR-TR-94-17, Microsoft Research, Microsoft Corporation, 1994.
11. Goldberg, D.E., "The Existential Pleasures of Genetic Algorithms". Genetic Algorithms in Engineering and Computer Science , (England: John Wiley and Sons Ltd., 1995) Ed. Winter, G., et al 23-31.
12. Holland, J. H., Adaptation in Natural and Artifical Systems, (Ann Arbor: The University of Michigan Press, 1975)
13. Larranga, P., et al., Genetic Algorithms Applied to Bayesian Networks,Proc. of the Applied Decision Technology Conf., 1995,pp283-302.
14. Bertoni, A., Dorigo, M., Implicit Parallelism in Genetic Algorithms. AI (61) 2, 1993, pp307-314.
15. Ben-Ari, M., Principles of Concurrent and Distributed Programming, (Hertfordshire: Prentice Hall International (UK) Ltd. ,1990)
16. Almeida, F., Garcia, F., Roda, J., Morales, D., Rodriguez C., A Comparative Study of two Distributed Systems: PVM and Transputers, Transputer Applications and Systems, (IOS Press, 1995) Ed. Cook, B., et al 244-258.