A Parallel Knowledge Discovery System for Customer Profiling

M. Coppola[2], P. Pesciullesi[1], R. Ravazzolo[1], and C. Zoccolo[1]

[1] University of Pisa, Dip. di Informatica, Via Buonarroti 2, 56127 Pisa, Italy
[2] Ist. di Scienza e Tecnologie dell'Informazione, CNR,
Via Moruzzi 1, 56124 Pisa, Italy

Abstract. We describe a parallel KDD architecture we are developing as part of an open-source based customer relationship management system, in the framework of the SAIB industrial research project. The design of the prototype, leveraging on the features of the ASSIST programming environment, results in a high-performance parallel data mining core, tightly integrated with parallel data management and interfaced to business standard technologies and systems.

1 Introduction

In this paper we describe the system architecture and the implementation of a parallel Knowledge Discovery in Databases (KDD) system we are developing for Customer Profiling in the framework of the SAIB project (System for Internet Banking Applications).

Developing a parallel KDD system is an interesting challenge in itself, and also with respect to the main focus of our research on parallel programming environments. Beside developing efficient parallel mining algorithms, a key issue is the degree of integration the system can reach, both internally, as simplicity and performance of the interaction between mining algorithms and parallel data management, and externally, as ease of cooperation with different software technologies. Industry standard languages and technologies like XML, Java, or component programming have to be exploited to integrate advanced parallel modules within larger applications.

We describe the efforts made so far toward this goal, also focusing on the advantages that a high-level parallel programming environment like ASSIST [1] can bring in designing a parallel KDD architecture. In Sect. 2 we introduce the SAIB research project and present the overall architecture of the system. Sect. 3 summarizes our past research and the features of the ASSIST environment. We describe in Sect. 4 the design of the parallel KDD system, and in Sect. 5 the implementation of the parallel mining primitives and a simple case study we are using to test system integration. Sect. 6 summarizes results and future work directions.

* This work has been supported by the SAIB Project on High-performance infrastructures for financial applications, funded by MIUR and leaded by SchlumbergerSEMA, owned by ATOS Origin, and by the Italian MIUR Strategic Project L.449/97-2000 on High-performance distributed enabling platforms.

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 381–390, 2004.

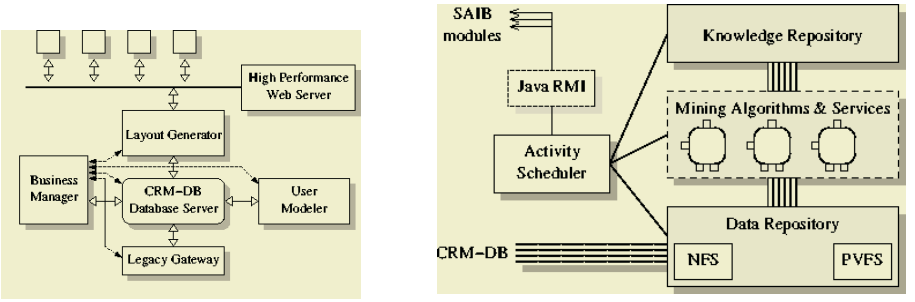

Fig. 1. (a) Overall Architecture of the SAIB system — (b) The UMS architecture.

2 A Data Mining Engine for the SAIB Project

SAIB is a large research project which brings together several Italian academic institutions and industrial partners in the effort of producing a flexible, open-source based Customer Relationship Management (CRM) solution for Internet Banking and Insurance.

The SAIB project has as essential goals to provide: (1) compatibility and cooperation with existing Hw/Sw infrastructures, in order to enhance their performance, (2) flexibility and programmability of the CRM solution, (3) privacy and security of end-user interaction, (4) multi-channel (e.g. kiosk, e-mail, mobile) and Internet-based interaction, (5) high performance customer profiling functionalities. The ultimate goal of the project goes beyond the Internet Banking solution, and it is to design a development environment for the broader class of CRM applications, including e.g. IT support systems for the public administration and call centers. In the SAIB system architecture (a partial overview is shown in Fig. 1a) a workflow interpreter (the Business Manager) executes a number of tasks, mostly triggered by local or remote user interaction. It controls a set of application modules which perform actual computation and interaction. The system can be easily tailored to different applications by adding new functional modules and new workflow programs.

Interchange of parameters and data among SAIB modules exploits industry standards like EJB interfaces, XML encoded Java RMI, and ODBC connection with a database server (the CRM-DB), which is a centralization point of the system. The Legacy Gateway module provides translation and integration of the Business Manager protocols to let SAIB cooperate with existing legacy systems.

Our KDD prototype is one of the main modules of the system, the User Modeler Server (UMS), providing static and dynamic customer profiling functions. We can roughly divide SAIB users into customers and administrative users. The UMS provides a restricted class of the administrative users (e.g. marketing analysts) with Knowledge Discovery and Data Mining services, in order to build customer profiles from the CRM-DB database. Knowledge models can then be deployed to the CRM main core, where they can be used by workflow programs to customize user interaction at different levels. Workflow programs can either

access information that has been downloaded to the main database, or issue on-line, per-user queries to the data mining engine.

The UMS thus performs both *batch* operations (heavyweight, but that do not condition the main business flow) and *on-line* ones, which are subject to near-real-time constraints. To avoid interference with the main system, and to allow higher performance to the parallel mining algorithms, the UMS operates primarily on data cached in its internal, parallel data management module, tightly coupled with the mining engine. The UMS controlling interface allows the user to load data from the CRM-DB, and to put back new information. The example we describe at the end of Sect. 5 is a simple customer segmentation and classification process. It is just one of the feasible applications of KDD to CRM, that range from user interface personalization to potential fraud detection.

We wanted the Data Mining engine of SAIB to be able to deal with databases of several Gigabytes in size, to distribute computation and I/O in parallel and to scale with available computing resources to higher performance and throughput. In HPC these goals are often hard to meet with a portable, high-level software design. On the contrary, the programming approach of the ASSIST environment allowed us to efficiently develop parallel application modules and high-performance libraries, merging them into a complex application.

3 The ASSIST Parallel Programming Environment

The adoption of a high-level parallel programming environment to develop the high-performance modules is one of the central assumptions of the SAIB project.

Our research in the field stems from the skeletons model [2], one in the class of structured parallel programming (SPP) models. SPP models, by describing the parallel semantics of programs in a high-level way, provide increased portability, ease of code reuse and application evolution with respect to low-level parallel programming approaches based on communication libraries. The approach is characterized by the (hierarchical) composition of modules with completely defined interfaces, each composition mapping to a set of known implementation templates. However, too strong constraints can make it difficult to develop complex and dynamically behaving applications.

With ASSIST [1, 3] we close in to component-based parallel programming. A program is an unrestricted graph of sequential and parallel modules interacting through data streams. Parallel modules (parmods) can express mixed data-parallel and task-parallel computations, can explicitly manage load balancing and non-determinism if needed, have an internal state, and can interface to external resources. Each parmod (or combination of parmods) is then easily used as a component of larger applications.

The design of the ASSIST model explicitly targets the needs of large applications over massively parallel platforms and Computational Grids, taking into account issues like dynamic resource adaptiveness, dependability, heterogeneity. Some of the corresponding features have already been implemented in the ASSIST environment at present time (we refer the reader to more specific

works like [4]), while the more general problem of devising a GRID programming model is being tackled in the framework of the ongoing national research project *"Grid.it"*.

In the KDD architecture we have implemented, we exploited several features of the ASSIST coordination language. Among them the support for *external objects* [1], that are object-like interfaces used from inside ASSIST code modules to access heterogeneous software resources. External objects can have their own run-time support, that must not interact with that of the application. In this case we have used the SMReference objects to manage large dynamic data structures in virtual shared memory, and we implemented the support for a parallel file system as an external object library for parallel and sequential modules.

4 Knowledge Discovery Architecture

The overall architecture of the KDD engine we have designed (Fig. 1b) is quite straightforward, based on four main modules providing

1. data management functionalities (the Data Repository, **DR**)
2. knowledge and meta-data management (the Knowledge Repository, **KR**),
3. a set of mining algorithms (**MA**s) and
4. a control interface, the **Activity Scheduler**.

The **Scheduler** interfaces to the rest of the SAIB system by means of Java RMI. Since it is not performance critical, the scheduler is actually implemented in Java, cooperating with the rest of the KDD engine by Java native methods, file system I/O and through the ASSIST program loader.

The scheduler accepts synchronous and asynchronous operation requests, enqueues them internally and manages the corresponding parallel programs and their results. Though the current SAIB architecture does not use it yet, concurrent operation execution with task priorities and dependencies can be dealt with in the scheduler. For testing purposes, a simple graphical Java front-end has been developed that connects to the scheduler and controls it.

The set of **MA modules** contains ASSIST parallel programs that perform actual mining tasks, and simpler ones used to manage the DR data (e.g. selection and sorting). We reused parallel mining applications designed in our previous research [5] and extended them as well with new functionalities. Taking advantage of the modular structure of ASSIST programs, we have evolved them so that they (1) interface to the DR module for most of the I/O, and (2) expose a common set of program interfaces (streams for data I/O, knowledge I/O and an XML-encoded file with running parameters).

This second requirement allows MAs to be viewed as a kind of software components within our system, decoupling their actual implementation from that of the mining engine as a whole. Adding new MA modules to the set known to the scheduler is simply a matter of defining their specific parameters as an XML schema. Moreover, standard conforming modules can be automatically composed into a larger ASSIST program to be compiled and run. This is still a work in progress to simplify and improve the performance of complex mining/validation processes that have to be executed routinely.

The **DR module** is implemented as an external object of ASSIST. It provides high performance I/O support for large files with simple record structure (the kind of regular data tables we have to host in a mining warehouse), a set of data types suitable to meet mining needs, and a mechanism of block-oriented views to allow parallel operation on the same file (a *dataset*, in the following) in a controlled way. It is a common choice to have data management functionalities integrated within the mining architecture. We aimed at a software layer that offered less overhead and more control on low-level issues w.r.t relational DBMS, while still providing a richer and more portable interface to data than working with flat files. For instance we use a block-oriented interface to allow explicit secondary memory management and concurrent operation within the MAs, since a large I/O grain is used anyway in the MA, and the waste of space is negligible.

Current DR implementation provides data types encoding floating-point numbers, nominal values from unordered list of labels, date values, booleans, unique keys and fixed-size uninterpreted raw data. We chose to have a fixed set of machine-dependent representations, to allow direct memory loading of data tables without translation, minimizing memory requirements and computational overhead in the algorithms. Special `UNKNOWN` values are provided for all types, and meta-data is kept linked with each dataset.

Most of these functions are provided as a library linked to the ASSIST generated code. A lower implementation layer moves data blocks in and out of each process memory, wrapping the actual file system layer. This design exploits parallel file system performance and bandwidth from within portable ASSIST programs, by allowing the data block engine to initiate concurrent data transfers across multiple I/O and processing nodes. The prototype is based on PVFS [6], and sequential UNIX file systems are supported too, including NFS, with lower I/O performance.

The **KR module** manages the knowledge produced by the mining algorithms, allowing to store, retrieve, refine knowledge models, and to track their history. Models are represented using the standard PMML 2.0 language [7], exploiting the PMML extension mechanism in a few cases where non-standard model semantics is needed. Each model is a PMML file containing the results of one or more mining algorithms, a link to the source dataset, and all the relevant meta-data and algorithmic parameter information. Models are interconnected so that a full KDD process can be designed and stored as a unit in the KR.

Knowledge models also have different states and attributes that condition their use inside the UMS, and their visibility in the whole SAIB system (e.g. a model has to be validated before its information is downloaded to the CRM-DB). A standard CVS server is used to store the actual PMML data, access to models being mediated by a custom server which performs additional controls and attribute caching. A client-side linked library provides XML parsing and serialization on models, as well as interface to the server processes.

5 Mining Primitives

The mining algorithms we have implemented are derived from our earlier experience in Data Mining with structured parallel programming environments [5].

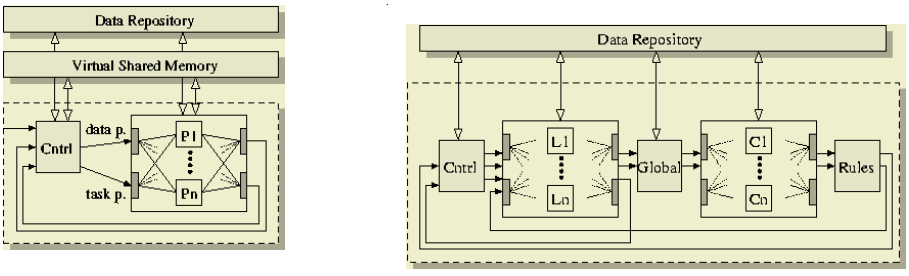

Fig. 2. (a) The parallel classifier. (b) Association-rule based clustering.

They have been extended, making use of the new features of ASSIST. We summarize here the basic UMS functions and the key points of their implementation.

We also show a few performance results measured on a cluster of 8 Pentium4/2GHz/512Mb RAM processing nodes with ATA UDMA disks, connected by a Gbit Ethernet switch and running Red Hat Linux 7.3.

Classification is performed with a decision tree induction algorithm. It uses the same score functions as the C4.5 classifier by Quinlan, but it currently works on nominal attributes only. Tree induction is a typical divide and conquer process, where a tree is built from the root (the whole input), at each node evaluating and partitioning the available data, each node expansion being independent from those happening on separate branches of the tree. Beside parallelism in the tree visit, large partitions call up for data-parallel decomposition of the splitting computation. Our prototype (Fig. 2a) keeps in the DR module data partitions associated with nodes, loading them on demand. The tree structure is local to the controller module, and statistic data are shared using SMReference external objects, thus the prototypes exploits a two-layer distributed memory hierarchy.

The expansion policy is given by a single process, controlling a parmod that performs all the computation. Expanding a node is a (possibly large) task, on which data-parallel, globally synchronized operations as well as task-parallel, concurrent ones are possible. Each task can result in more tasks to be produced (new nodes added to the tree) or in a fully sequential sub-computation (a complete subtree is generated down to the leaves). For the sake of conciseness, we disregard the fact that further decomposition happens of the parallel activities into evaluation and splitting steps.

After a first phase of the execution where the data parallel decomposition is used (current prototype actually employs this strategy only for the root node), we switch to the task-parallel behaviour, and below a certain node size to sequential computation. Dynamic load balancing in the task parallel case is guaranteed by the ASSIST support. The expansion policy, based on node size, determines the computation switch points and the relative priorities of different nodes.

ASSIST features allowed us to improve the program structure reported in [5] following the idea outlined in [1], i.e dynamically mixing data and task parallelism in the same high-level program. W.r.t. [5] we exploited the flexibility of

the ASSIST parmod construct to express at the coordination level the different but correlated functionalities of the Conquer module (e.g. task parallel expansion, data parallel counting, and sorting). Stream guards controlled by shared variables make it clear and manageable the transition among the different behaviors. Data-parallel expansion allows to exceed the main-memory limits of a single processing node.

W.r.t. the solution in [1] we exploit a two-level distributed memory hierarchy (shared memory and parallel file system) using two kinds of external object. However, data-parallel expansion is still limited to the first node. We are currently developing the adaptive behaviour of the classifier to allow tuning of the transition from data to task parallelism, which is needed for large datasets, and to let the application deal with numeric attributes. We think that these improvements will also benefit performance on medium-size datasets. Figure 3a shows the speedup in such a case (sequential completion time is 43 seconds).

Association Rules are computed by an Apriori-like algorithm. Its parallel version is based on the partitioning method and requires two phases, each one performed in parallel on separate partitions of the input. As in [5], the input dataset is scanned two times in full, load-balancing being guaranteed by the on-demand distribution of ASSIST, but here we exploit the DR module to dispatch partitions from the hosting nodes to the requesting ones.

The partitioned method has a good parallel speed-up and it is scalable w.r.t. the number of transactions in the dataset. Fig. 3b shows almost linear speed-up for a medium-size, synthetic dataset (1.2M transactions of av. length 30).

Clustering is developed around the association rules module. We use a notion of clustering derived from [8], grouping together records that satisfy the longest, most popular association rule of a dataset. To produce more clusters, rules are mined again and again on unclustered records until a threshold on support or a prefixed number of clusters is reached.

The parallel implementation reuses the association rule main modules, the local tree build and the counting parmods, exploiting additional streams and guarded channels to control the iteration process, and to continuously reorganize a temporary copy of the input dataset. Figure 2b shows a simplified representation of the clustering application, including streams that carry on block indexes, frequent itemset information and cluster defining rules.

The rearranging strategy employed by the L_i modules during local frequent set search accumulates already clustered records into a "black list" of data blocks to be subsequently skipped, in order to enhance locality and progressively reduce the amount of I/O. Active blocks saved on disk are reassigned to any waiting process, ensuring proper load balancing. After the first phase, a global counting phase follows and an association rule is selected as a cluster definition. Clustered records are then discarded by the rearranging policy while searching for the next best rule.

Filters to perform basic manipulations on datasets can be implemented within the DR interface of an algorithm, or as stand-alone parallel programs that are run by the Scheduler like any MAs. In the first class there are field and record

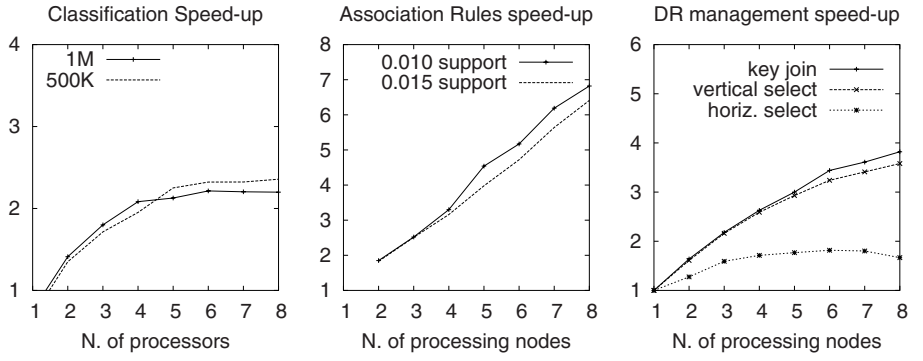

Fig. 3. Speed-up tests with 8 I/O nodes and N processing nodes. (a) Tree induction from 4M instances of the LED database, 20 attributes, 4% noise, 10 classes; switch to sequential computation at 500K, 1M node size. (b) Association Rule computation: 156Mb dataset, 1.2M trans., 661s and 1468s sequential time. (c) Simple key join, vertical and horizontal select DR routines: 2Gb dataset; peak data transfer bandwidth achieved is in the 86Mb/s – 141Mb/s range.

selections from a dataset, while the second class contains more complex operation like k-way merge-sorting, merging of datasets w.r.t. a common key attribute, summarization operators. In Fig. 3c we see the speedup obtained by three simple filter programs (key join, column and record selection) operating on datasets of 1 – 2Gbytes. Here communication bandwidth is the main limit to the speedup. Knowledge validation functions are also implemented as MAs. As an example, current UMS prototype provides functions to compute confusion matrices, in order to evaluate classification or clustering models.

Case Study. We have chosen a simplified case study to check the overall system design, showing correct interoperability of the UMS basic blocks. The target is to customize commercial advertising by developing a set of customer profiles, and a classifier that allows us to assign new customers to a base profile, on the ground of the limited amount of information that is initially available.

The initial data comes as a special purpose table stored within the CRM-DB, one record per customer, containing a large set of summarized attributes about customer behaviour. The input data is periodically imported to a DR dataset. New users may either be missing from this dataset, or correspond to records with many missing values.

The knowledge extraction process is based on the distinction of customer attributes into *factual* ones, that identify the user and do not change quickly over time, and *behavioral* ones, historical data about user interaction which are derived from transactional databases, and reflect evolving customer's commercial attitude. A process based on a similar data model is reported in [9].

We start the mining process extracting clusters from the dataset, by applying the association-rule-based clustering algorithm to the available behavioral data of customers. We try to identify classes of customers with similar habits, e.g. using the same bank services in the same period of time. After expert's validation, the set of cluster-defining rules is turned into a class label definition for all user

with sufficient behavioral information. The second step is to infer a classification tree for customers, modeling the class label in terms of the factual attributes. After another validation step and possibly more iterations of the process, we can classify new users using the tree even if some of the factual attributes, usually available, are missing. Clearly, the distinction of attributes into factual and behavioral is not dogmatic, and trying different attribute sets for both process steps is part of the experimental part of the KDD process.

6 Conclusions and Future Work

We have implemented a parallel KDD architecture, written in Java, C++ and the ASSIST coordination language, with support for parallel I/O and mining algorithms, that is integrated with industry standard protocols and provides a Java RMI control interface. In the process we exploited a modular approach to integrate sequential and parallel code, reusing and modifying existing parallel kernels to integrate them into a single system. As a side effect of the project we have integrated in the ASSIST environment support for the PVFS parallel file system, using the abstraction of external objects.

The resulting design distinguishes from distributed mining frameworks like Papyrus [10]. Our KDD prototype fits in a general framework for high performance application development, not limited in scope to data mining. As a consequence, we designed a block-oriented, shared-memory like data distribution layer, instead of a sophisticated data transport layer like Papyrus' one.

Moreover, we aim at efficiently exploiting the processing power of Beowulf clusters. With respect to existing distributed mining systems relying on coordination of many independent sequential mining engines, like [11], our approach can be applied to a broader set of mining tasks, and it is much less influenced by the memory and computing power constraints of any single machine.

Work is still in progress to improve the different components of the system, especially to enrich the set of mining functionalities and to improve the management of meta-data within the KDD process. Tests are ongoing according to the simple KDD process outlined to verify the degree of integration with the SAIB CRM solution.

Our system is not constrained to a specific class of computing platforms. The current testbed is a small dedicated cluster of tightly coupled machines, but the ASSIST environment allows us to seamlessly run applications on clusters, LANs and WANs, as well as on grids and heterogeneous clusters (w.r.t. CPU architecture, O.S., and performance). While not every combination of different settings is already supported, the current implementation of ASSIST and of the KDD system makes it feasible to run mining algorithms on local networks with a common CPU architecture.

Moving to more distributed platforms opens up new research issues about the scheduling of parallel activities, and about the forecast of performance and scalability of the resulting KDD engine, as bandwidth and latency constraints affect the efficiency of many of the mining tasks. The problem in perspective merges with our current research on high-level Grid Programming environments.

Acknowledgments

We wish to thank SchlumbergerSEMA S.p.A., M. Vanneschi and M. Danelutto, and all the people of the Parallel Architectures Group in Pisa, who work on various aspects of the SAIB project or contribute to the development of ASSIST: M. Aldinucci, S. Campa, P. Ciullo, S. Magini, A. Paternesi, A. Petrocelli, E. Pistoletti, L. Potiti, M. Torquati and P. Vitale.

References

1. Vanneschi, M.: The programming model of ASSIST, an environment for parallel and distributed portable applications. Parallel Computing **28** (2002) 1709–1732
2. Cole, M.: Algorithmic Skeletons: Structured Management of Parallel Computations. Research Monographs in Parallel and Distributed Computing, Pitma (1989)
3. Aldinucci, M., Campa, S., Ciullo, P., Coppola, M., Magini, S., Pesciullesi, P., Potiti, L., Ravazzolo, R., Torquati, M., Vanneschi, M., Zoccolo, C.: The Implementation of ASSIST, an Environment for Parallel and Distributed Programming. In Kosch, H., László Böszörményi, Hellwagner, H., eds.: Euro-Par 2003: Parallel Processing. Number 2790 in LNCS (2003) 712–721
4. Aldinucci, M., Campa, S., Magini, S., Pesciullesi, P., Potiti, L., Ravazzolo, R., Torquati, M., Zoccolo, C.: Targeting Interoperability and Heterogeneous Architectures in ASSIST. To appear in Proc. of the Euro-Par 2004 Int.Conf., Pisa (2004)
5. Coppola, M., Vanneschi, M.: High-Performance Data Mining with Skeleton-based Structured Parallel Programming. Parallel Computing, special issue on Parallel Data Intensive Computing **28** (2002) 793–813
6. Carns, P.H., Ligon, W.B. III., Ross, R.B., Thakur, R.: PVFS: A Parallel File System For Linux Clusters. In: Proc. of the 4th Annual Linux Showcase and Conference. (2000) 317–327
7. The Data Mining Group: Pmml 2.0 specification. http://www.dmg.org/pmml-v2-0.html (2003)
8. Kosters, W.A., Marchiori, E., Oerlemans, A.A.J.: Mining clusters with association rules. In Hand, D., Kok, J., Berthold, M., eds.: Advances in Intelligent Data Analysis: 3rd Int.l Symp., IDA-99. Volume 1642 of LNCS. (1999) 39–50
9. Adomavicius, G., Tuzhilin, A.: Using data mining methods to build customer profiles. Computer **34** (2001) 74–82
10. Bailey, S., Creel, E., Grossman, R., Gutti, S., Sivakumar, H.: A High Performance Implementation of the Data Space transfer Protocol (DSTP). In Zaki, M.J., Ho, C.T., eds.: Large-Scale Parallel Data Mining. Volume 1759 of LNAI. Springer (1999) 55–64
11. Musicant, D., Celis, S.: Weka-parallel: Machine learning in parallel. Technical report, Carleton College of Computer Science (2000)