Using Mining@Home for Distributed Ensemble Learning

Eugenio Cesario[1], Carlo Mastroianni[1], and Domenico Talia[1,2]

[1] ICAR-CNR, Italy
{cesario,mastroianni}@icar.cnr.it
[2] University of Calabria, Italy
talia@deis.unical.it

Abstract. Mining@Home was recently designed as a distributed architecture for running data mining applications according to the "volunteer computing" paradigm. Mining@Home already proved its efficiency and scalability when used for the discovery of frequent itemsets from a transactional database. However, it can also be adopted in several different scenarios, especially in those where the overall application can be divided into distinct jobs that may be executed in parallel, and input data can be reused, which naturally leads to the use of data cachers. This paper describes the architecture and implementation of the Mining@Home system and evaluates its performance for the execution of ensemble learning applications. In this scenario, multiple learners are used to compute models from the same input data, so as to extract a final model with stronger statistical accuracy. Performance evaluation on a real network, reported in the paper, confirms the efficiency and scalability of the framework.

1 Introduction

The global information society is a restless producer and exchanger of huge volumes of data in various formats. It is increasingly difficult to analyze this data promptly, and extract from it the information that is useful for business and scientific applications. Fortunately, the notable advancements and the advent of new paradigms for distributed computing, such as Grids, P2P systems, and Cloud Computing, help us to cope with this data deluge in many scenarios.

Distributed solutions can be exploited for several reasons: (i) data links have larger bandwidths than before, enabling the assignment of tasks and the transmission of related input data in a distributed scenario; (ii) data caching techniques can help to reuse data needed by different tasks, (iii) Internet computing models such as the "public resource computing" or "volunteer computing" paradigm facilitate the use of spare CPU cycles of a large number of computers.

Volunteer computing has become a success story for many scientific applications, as a means for exploiting huge amount of low cost computational resources with a few manpower getting involved. Though this paradigm is clearly suited for the exploitation of decentralized architectures, the most popular volunteer computing platform available today, BOINC [2], assigns tasks according to a

A. Hameurlain et al. (Eds.): Globe 2012, LNCS 7450, pp. 100–111, 2012.

centralized strategy. Recent work, though, showed that the public computing paradigm can be efficiently combined with decentralized solutions to support the execution of costly data mining jobs that need to explore very large datasets. These reasons leaded to the design of the MINING@HOME architecture. In [11], simulation experiments showed that the architecture is able to solve the Closed Frequent Itemsets problem. After these early simulations that showed the benefits of the proposed approach, we worked to provide a full implementation of the framework and here we show that it is capable for the efficient execution of different data mining applications in a distributed scenario.

The main contribution of this work is the description of the MINING@HOME system architecture and implementation, and the evaluation of its performance when running "ensemble learning" applications in a real scenario. The ensemble learning approach combines multiple mining models together instead of using a single model in isolation [4]. In particular, the "bagging" strategy consists of sampling an input dataset multiple times, to introduce variability between the different models, and then extracting the combined model with a voting technique or a statistical analysis. MINING@HOME was profitably adopted to analyze a transactional dataset containing about 2 million transactions, for a total size of 350 MB. To run the application, the volunteer paradigm strategy is combined with a super-peer network topology that helps to exploit the multi-domain scenario adopted for the experiments.

The reminder of the paper is organized as follows: Section 2 presents the architecture, the involved protocols and the implementation of MINING@HOME . Section 3 discusses the ensemble learning strategy. Section 4 illustrates the scenario of the experiments and discusses the main results. Finally, Section 5 discusses related work and Section 6 concludes the paper.

2 Architecture and Implementation of Mining@home

As mentioned, the MINING@HOME framework was introduced in [11] to solve the problem of finding closed frequent itemsets in a transactional database. The system functionality and performance were only evaluated in a simulated environment. After that, MINING@HOME was fully implemented and was made able of coping with a number of different data analysis scenarios involving the execution of different data mining tasks in a distributed environment. The architecture of the MINING@HOME framework distinguishes between nodes accomplishing the mining task and nodes supporting data dissemination. In the first group:

- the **data source** is the node that stores the data set to be read and mined.
- the **job manager** is the node in charge of decomposing the overall data mining application in a set of independent tasks. This node produces a *job advert* document for each task, which describes its characteristics and specifies the portion of the data needed to complete the task. The job manager is also responsible for the collection of results.
- the **miners** are the nodes available for job execution. Assignment of jobs follows the "pull" approach, as required by the volunteer computing paradigm.

Data exchange and dissemination is done by exploiting the presence of a network of super-peers for the assignment and execution of jobs, and adopting caching strategies to improve the efficiency of data delivery. Specifically:

- **super peer** nodes constitute the backbone of the network. Miners connect directly to a super-peer, and super-peers are connected with one another through a high level P2P network.
- **data cachers** nodes operate as data agents for miners. In fact, data cachers retrieve input data from the data source or other data cachers, forward data to miners and store data locally to serve miners directly in the future.

The super-peer network allows the queries issued by miners to rapidly explore the network. The super-peer approach is chosen to let the system support several public computing applications concurrently, without requiring that each miner knows the location of the job manager and/or of the data cachers. Super-peers can also be used as rendezvous points that match job queries issued by miners with job adverts generated by the job manager.

The algorithm is explained here (see Figure 1). Firstly, the job manager partitions the data mining application in a set of tasks that can be executed in parallel. For each task, a "job advert" specifies the characteristics of the task to be executed and the related input data. An available miner issues a "job query" message to retrieve one of these job adverts. Job queries are delivered directly to the job manager, if it is possible. If the location of the latter is not known, job queries can travel the network through the super-peer interconnections (messages labeled with number 1 in the figure). When a job advert is found that matches the job query, the related job is assigned to the miner (message 2 in the figure). The miner is also informed, through the job advert, about the data that it needs to execute the job. The required input data can be the entire data set stored in the data source, or a subset of it.

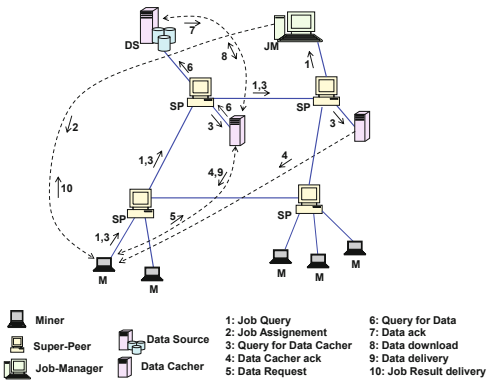

Fig. 1. Architecture of Mining@home

The miner does not download data directly from the data source, but issues a query to discover a data cacher (message 3). This query can find several data cachers, each of which sends an ack to the miner (message 4). After a short time interval, the miner selects the most convenient data cacher according to a given strategy (message 5), and delegates to it the responsibility of retrieving the required data. The data cacher issues a "data request" (message 6) to discover the data source or another data cacher that has already downloaded the needed data. The data cacher receives a number of acks from available data cachers (message 7), downloads the data from one of those (message 8), stores the data, and forwards it to the miner (message 9). Now the miner executes the task and, at its completion, sends the results to the job manager (message 10).

The algorithm can be used in the general case in which the location of job manager, data source and data cachers is unknown to miners and other data cachers. In ad hoc scenarios the algorithm can be simplified. Specifically, if the location of data source and data cachers are known, a job query (message 1) can be delivered directly to the job manager, instead of traveling the network, and messages 3-4 and 6-7 become unnecessary. Such simplifications are adopted for the experimental evaluation discussed in Section 4.

The MINING@HOME prototype has been implemented in Java, JDK 1.6. As depicted in Figure 1, the framework is built upon five types of nodes: job manager, data source, data cacher, super-peer and miner. Each node is multi-threaded, so that all tasks (send/receive messages, retrieve data, computation) are executed concurrently. Each miner exploits a *Mining Algorithm Library*, i.e., a code library containing the algorithms corresponding to the mining tasks.

3 Ensemble Learning and Bagging

Ensemble learning is a machine learning paradigm where multiple learners are trained to solve the same problem. In contrast to ordinary machine learning approaches, which try to learn one model from training data, ensemble methods build a set of models and combine them to obtain the final model. In a classification scenario, an ensemble method constructs a set of *base classifiers* from training data and performs classification by taking a vote on the predictions made by each classifier. As proved by mathematical analysis, ensemble classifiers tend to perform better (in terms of error rate) than any single classifier [12]. The basic idea is to build multiple classifiers from the original data and then aggregate their predictions when classifying unknown examples.

Bagging, also known as "bootstrap aggregating", is a popular ensemble learning technique [5]. Multiple training sets, or *bootstrap samples*, are sampled from the original dataset. The samples are used to train N different classifiers, and a test instance is labeled by the class that receives the highest number of votes by the classifiers. A logical view of the bagging method is shown in Figure 2. Each bootstrap sample has the same size as the original dataset. Since sampling is done with replacement, some instances may appear several times in the same bootstrap sample, while others may not be present. On average, a bootstrap

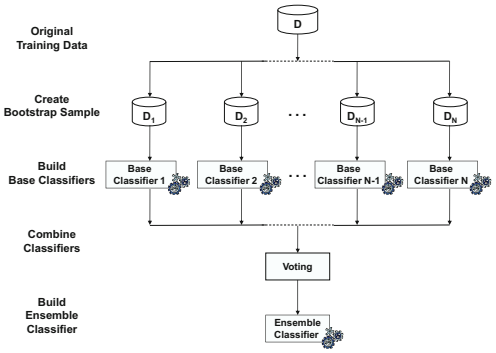

Fig. 2. A logical view of the bagging technique

sample D_i contains approximatively 63% of the original training data. In fact, if the original dataset contains n instances, the probability that a specific instance is sampled at least once is: $1 - (1 - 1/n)^n \rightarrow 1 - 1/e \simeq 0.631$, where the approximation is valid for large values of n. This implies that two different samples share, on average, about $0.631 \cdot 0.631 \approx 40\%$ of the n instances.

An application implementing the bagging technique can naturally exploit the MINING@HOME system. The described scenario matches the two main conditions that must hold in order to profitably exploit the features of MINING@HOME :

1. *Base Classifiers Are Independent.* Each base classifier can be mined independently from each other. Thus, it is possible to have a list of mining tasks to be executed, each one described by a distinct job descriptor. This fits the MINING@HOME architecture: each available miner is assigned the task of building one base classifier from a bootstrap sample of data, and at the end of execution the discovered classification model is transmitted to the job manager. Then, the miner may give its availability for a new job.

2. *Data Can Be Re-Used.* As mentioned before, in general different jobs need overlapping portions of input data, which is the rationale for the presence of distributed cache servers. After being assigned a job, the miner asks the input data to the closest data cacher, which may have already downloaded some of this data to serve previous requests. The data cacher retrieves only the missing data from the data source, and then sends the complete bootstrap sample to the miner. Of course, this leads to save network traffic and to a quicker response from the data cacher.

4 Experimental Evaluation

The performance of the MINING@HOME framework has been evaluated on a classification problem tackled with the bagging technique. We deployed the framework in a real network composed of two domains connected through a Wide Area

Network, as depicted in Figure 3. Each node runs an Intel Pentium 4 processor with CPU frequency 1.36GHz and 2GB RAM. The average inter-domain transfer rate is $197KB/s$, while the average intra-domain transfer rates are $918KB/s$ and $942KB/s$, respectively. The experiments were performed in a scenario where the job manager builds 32 base classifiers, by exploiting the bagging technique, on a transactional dataset D. The application proceeds as follows. The job manager builds a *job list*, containing the descriptions of the jobs that must be assigned to available miners. Each job is a request of building a J48 base classifier from a specific bootstrap sample. When all the jobs are executed, the job manager collects the extracted base classifiers and combines them to produce the final ensemble classifier.

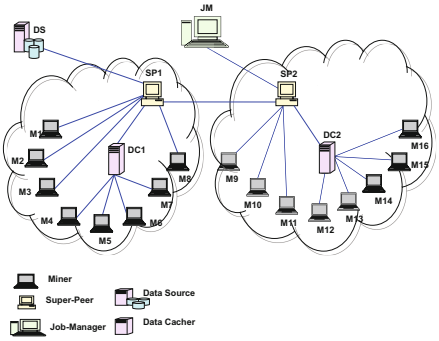

Fig. 3. Network architecture for the MINING@HOME experiments

The input dataset D is a subset of the *kddcup99* dataset[1]. The dataset, used for the KDD'99 Competition, contains a wide amount of data produced during seven weeks of monitoring in a military network environment subject to simulated intrusions. The input dataset used for the experiments is composed of 2 million transactions, for a total size of 350 MB.

Our evaluation followed two parallel avenues: we approximated performance trends analytically, for a generic scenario with N_D domains, and at the same time we compared experimental data to analytical predictions for the specific scenario described before. We adopted the techniques presented in [9] for the analysis of parallel and distributed algorithms. Let T_s be the *sequential execution time*, i.e., the time needed by a single miner to execute all the mining jobs sequentially, and T_o the *total overhead time* (mostly due to data transfers) experienced when the jobs are distributed among multiple miners. The total time spent by all the processors to complete the application can be expressed as:

$$nT_p = T_s + T_o \tag{1}$$

in which n is the number of miners and T_p is the parallel execution time when n miners are used in parallel. The speedup S - defined as the ratio between

[1] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`

T_s and T_p - is then $S = \frac{T_s}{(T_s+T_o)/n} = \frac{nT_s}{T_s+T_o}$, and the efficiency E - defined as the ratio between the speedup and the number of miners - can be expressed as $E = \frac{1}{1+T_o/T_s}$. Therefore, the efficiency of the parallel computation is a function of the ratio between T_o and T_s: the lower this ratio, the higher the efficiency.

Let us start examining T_o. The total overhead time comprises the time needed to transfer data from the data source to data cachers and from these to miners. Both types of download are composed of a start up time (needed to open the connection and start the data transfer) and a time that is proportional to the amount of transferred data. Start up times are negligible with respect to the actual transfer time, therefore an approximation for T_o, in a scenario with N_D domains and N_{DC} data cachers, is:

$$T_o = \frac{|D|}{R_{DS}} \cdot N_{DC} + \sum_{i=1}^{N_D} (\frac{f \cdot |D|}{R_i} \cdot N_i) \tag{2}$$

where $|D|$ is the input data set size, R_{DS} is the average rate at which data is downloaded from the data source to a data cacher, R_i is the download rate from a data cacher to the local miners within the *i-th* domain, and N_i is the number of jobs assigned to the *i-th* domain. The expression of the first term derives from the necessity of delivering the whole dataset, in successive data transfers, to all the data cachers. On the other hand, for any of the N_i jobs that are executed under domain i, a fraction f (with $f \simeq 0.63$) of the dataset is downloaded by the miner from the local data cacher, which explains the second term.

Notice that T_o increases linearly with the dataset size $|D|$. On the other hand, the number of data cachers N_{DC} influences the two terms of expression (2) in different ways. The first term is proportional to N_{DC}, while the second term is inversely proportional to N_{DC} in an implicit fashion: when more data cachers are deployed on the network, possibly closer to miners, the intra-domain transfer rates R_i increase, and then the value of the second term decreases. Our experiments showed that the best choice is to have one data cacher per domain: this allows intra-domain transfer rates to be increased but at the same time avoids transmission of data to more data cachers than necessary, which would increase the first term in (2). This is the choice taken for our scenario, as shown in Figure 3. It is also possible to predict the impact of the number of miners. When two or more miners request data from the local data cacher at the same time, the intra-domain transfer rate R_i tends to decrease, because the uplink bandwidth of the data cacher is shared among multiple data connections. Since the probability of concurrent downloads increases when more miners are active in the same domain, the overall value of T_o tends to increase with the number of miners. This aspect will be examined in the comments to the experiments.

As opposed to T_o, the sequential time T_s does not depend on the network configuration, but of course it depends on the dataset size $|D|$. Specifically, the relationship between $|D|$ and T_s reflects the time complexity of the J48 algorithm, which is $O(|D| * ln|D|)$ [13].

Figure 4 reports the values of T_s and T_o measured in a scenario with two domains, one data cacher per domain, when varying the dataset size. The three

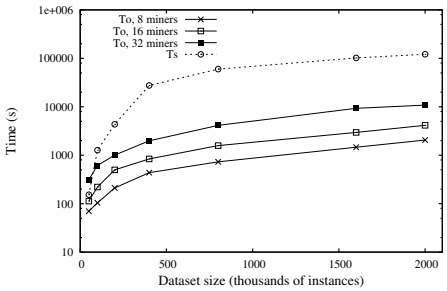

Fig. 4. Sequential time T_s and overhead time T_o vs. the size of the dataset, using 8, 16 and 32 miners

curves for T_o are obtained in the cases that 8, 16 and 32 miners are available to execute the 32 jobs. Of course, the values of T_s do not depend on the number of miners, as T_s is simply the sum of job computation times. The log scale is needed to visualize values that are very different from each other. The trends of T_s and T_o follow, quite closely, the theoretical prediction. This is shown in Figure 5, which compares the experimental values of T_s to the theoretical J48 complexity, and the experimental values of T_o, obtained with 16 miners, to those obtained with expression (2). While the gap between the two curves of T_o is small for any value of $|D|$, a larger discrepancy is observed for T_s when the dataset contains less than 500,000 instances, but this discrepancy tends to vanish for larger datasets. This is compatible with the fact that the expression $O(|D| * ln|D|)$ for the J48 theoretical complexity reflects an asymptotic behavior.

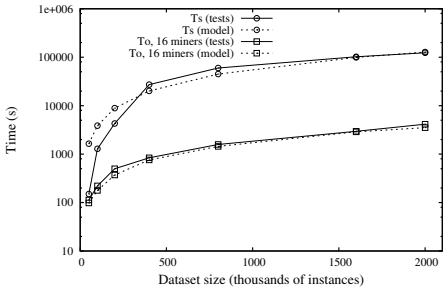

Fig. 5. Sequential time T_s and overhead time T_o (obtained with 16 miners) vs. the size of the dataset. Experimental results are compared to the theoretical prediction.

In Figure 4 it is also interesting to notice that the ratio T_o/T_s notably decreases in the first part the curve, which, as discussed before, is a sign that the efficiency of the architecture increases with the problem size. However, when the number of instances is higher than 500 thousands, the gap between T_s and T_o becomes stable, which in a graph with log scale means that the corresponding

ratio becomes stable. To make this clearer we reported, in Figure 6, the relative weighs of T_s and T_o. When the dataset size is moderate, the time spent for data transmission is comparable or even higher that the computation time, therefore the distributed solution is not convenient. When the size increases, however, the weigh of T_o becomes smaller, until it converges to very small values, below 10% of the total time. This is essential to justify the adoption of a distributed solution: if the overhead time were predominant, the benefit derived from the parallelization of work would not compensate the extra time needed to transfer data to remote data cachers and miners. It is also observed that the weigh of T_o increases when the number of available miners increases, due to the impact of concurrent downloads of multiple miners from the same data cachers.

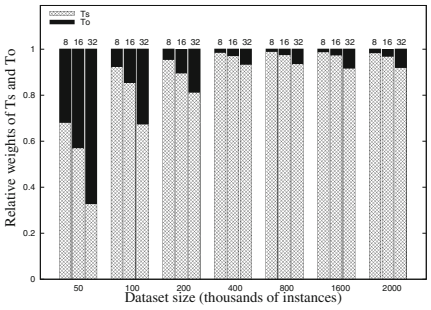

Fig. 6. Relative weights of sequential time T_s and overhead time T_o vs. the size of the dataset, using 8, 16 and 32 miners

To better analyze the last aspect, in Figure 7(a) we report the turnaround time (from the time the application is started to the completion time of the last job) vs. the number of available miners, which are equally partitioned between the two domains, except the case in which there is only one miner. Results are reported for three different sizes of $|D|$, and are plotted in a log scale due to the wide range of obtained values. The trend vs. the number of miners is the same at a first sight, but after a better look the system turns out to scale better when the problem is bigger. When processing 100,000 instances, the turnaround time decreases from 1,380 seconds with one miner to 160 seconds with 16 miners, but then nearly stabilizes to 145 seconds with 32 miners. With a dataset of 2 million instances, the turnaround time goes from about 34 hours with 1 miner to about 150 minutes with 16 miners and then continues to decrease to about 84 minutes with 32 miners. This indicates that using more and more miners can be efficient when the problem is big, but it is nearly useless – or even detrimental for the necessity of administrating a bigger system – when the problem size is limited. This is an index of good scalability properties, since scalable systems can be defined as those for which the number of workers that optimizes the performance increases with the problem size [9]. In Figure 7(b) we report the speedup, i.e., the ratio of the turnaround time obtained with a single node to

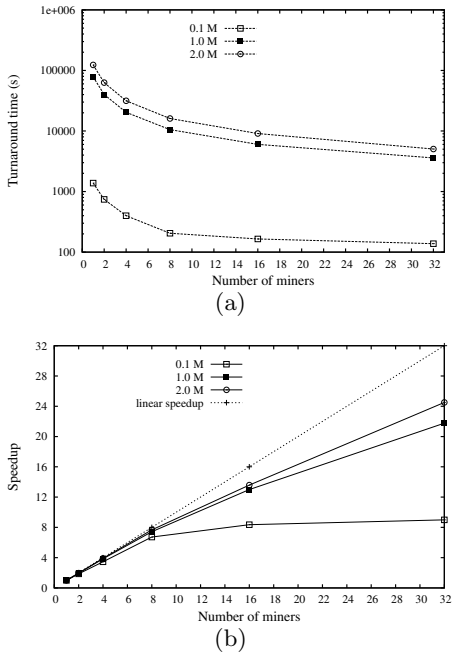

Fig. 7. Turnaround time (a) and speedup (b) vs. the number of available miners, for different values of the dataset size (1 M = 1 million instances)

the turnaround time computed with n nodes. It is clear that the speedup index saturates soon when the dataset is small, while the trend is closer to the optimal one (i.e., linear, shown for reference) as the dataset size increases.

5 Related Work

So far, the research areas of Distributed Data Mining and public resource computing have experienced little integration. The volunteer computing [1] paradigm has been exploited in several scientific applications (i.e., Seti@home, Folding@home, Einstein@home), but its adoption for mining applications is more challenging. The two most popular volunteer computing platforms available today, *BOINC* [2] and *XtremWeb* [6,8], are especially well suited for CPU-intensive applications but are somewhat inappropriate for data-intensive tasks, for two main reasons. First, the centralized nature of such systems requires all data to be served by a group of centrally maintained servers. Consequently, any server in charge of job assignment and data distribution is a clear bottleneck and a single point of failure for the system. Second, the client/server data distribution scheme does not offer valuable solutions for applications in which input data files can be initially stored in distributed locations or may be reused by different workers.

Some approaches to overcome such limitations have been recently proposed. In [7] and [11] the authors analyze, through a simulation framework, a volunteer computing approach that exploits decentralized P2P data sharing practices. The approach differs from the centralized BOINC architecture in that it seeks to integrate P2P networking directly into the system, as job descriptions and input data are provided to a P2P network instead of being directly delivered to the client. In particular, the application scenario discussed in [7] concerns the analysis of gravitational waveforms for the discovery of user specified patterns that may correspond to "binary stars", while [11] copes with problem of identifying closed frequent itemsets in a transactional dataset. So far, the analysis has only been performed in a simulation environment.

To the best of our knowledge, MINING@HOME is the first fully implemented public resource computing framework that executes data mining tasks over distributed data. In particular, its architecture is data-oriented because it exploits distributed cache servers for the efficient dissemination and reutilization of data files. The protocols and algorithms adopted by MINING@HOME are general and can be easily adapted to a wide set of distributed data mining problems. Not only this kind of solution can improve the performance of public computing systems, in terms of efficiency, flexibility and robustness, but also it can enlarge the use of the public computing paradigm, in that any user is allowed to define its own data mining application and specify the jobs that will be executed by remote volunteers, which is not permitted by BOINC.

More in general, several distributed data mining algorithms and systems have been proposed. In [3], a scalable and robust distributed algorithm for decision tree induction in distributed environments is presented. In order to achieve good scalability in a distributed environment, the proposed technique works in a completely asynchronous manner and offers low communication overhead. A distributed meta-learning technique is proposed in [10], where knowledge probing is used to extract descriptive knowledge from a black box model, such as a neural network. In particular, probing data is generated using various methods such as uniform voting, trained predictor, likelihood combination, etc. Differently from the classical meta-learning, the final classifier is learned from the probing data.

6 Conclusions

The public resource computing paradigm has proved useful to solve complex large problems in computational science areas, although it has not been used for data mining. In this paper we presented a software system, called MINING@HOME , which exploits that paradigm to implement large-scale data mining applications in a decentralized infrastructure. The developed system can be profitably used in the internet for mining massive amount of data available in remote Web sites or geographically dispersed data repositories. We evaluated the system and its performance on a specific use case, in which classification of instances is driven by the use of ensemble learning techniques. The system can be used for the execution of other mining applications, when these can be

decomposed in smaller jobs and can exploit the presence of distributed data cachers. Further applications of MINING@HOME are currently under investigation.

Acknowledgments. This research work has been partially funded by the MIUR projects FRAME (PON01_02477) and TETRIS (PON01_00451).

References

1. Anderson, D.P.: Public computing: Reconnecting people to science. In: Proceedings of Conference on Shared Knowledge and the Web, Madrid, Spain, pp. 17–19 (2003)
2. Anderson, D.P.: Boinc: A system for public-resource computing and storage. In: GRID 2004: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID 2004), Washington, DC, USA, pp. 4–10 (2004)
3. Bhaduri, K., Wolff, R., Giannella, C., Kargupta, H.: Distributed decision tree induction in peer-to-peer systems (2008)
4. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
5. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
6. Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Neri, V., Lodygensky, O.: Computing on large-scale distributed systems: Xtrem web architecture, programming models, security, tests and convergence with grid. Future Generation Computer Systems 21(3), 417–437 (2005)
7. Cozza, P., Mastroianni, C., Talia, D., Taylor, I.: A Super-Peer Model for Multiple Job Submission on a Grid. In: Lehner, W., Meyer, N., Streit, A., Stewart, C. (eds.) Euro-Par Workshops 2006. LNCS, vol. 4375, pp. 116–125. Springer, Heidelberg (2007)
8. Fedak, G., Germain, C., Neri, V., Cappello, F.: Xtremweb: A generic global computing system. In: Proceedings of the IEEE Int. Symp. on Cluster Computing and the Grid, Brisbane, Australia (2001)
9. Grama, A.Y., Gupta, A., Kumar, V.: Isoefficiency: Measuring the scalability of parallel algorithms and architectures. IEEE Concurrency 1, 12–21 (1993)
10. Guo, Y., Sutiwaraphun, J.: Probing Knowledge in Distributed Data Mining. In: Zhong, N., Zhou, L. (eds.) PAKDD 1999. LNCS (LNAI), vol. 1574, pp. 443–452. Springer, Heidelberg (1999)
11. Lucchese, C., Mastroianni, C., Orlando, S., Talia, D.: Mining@home: Towards a public resource computing framework for distributed data mining. Concurrency and Computation: Practice and Experience 22(5), 658–682 (2010)
12. Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Pearson International Edition (2006)
13. Witten, I.H., Frank, E.: Data mining: practical machine learning tools and techniques with Java implementations. Morgan Kaufmann (2000)