Inhambu: Data Mining Using Idle Cycles in Clusters of PCs

Hermes Senger[1], Eduardo R. Hruschka[1], Fabrício A.B. Silva[1],
Liria M. Sato[2], Calebe P. Bianchini[2], Marcelo D. Esperidião[2]

[1] Universidade Católica de Santos (UniSantos)
R. Dr. Carvalho de Mendonça, 144
11070-906 Santos-SP, Brazil
{senger, erh, fabricio}@unisantos.br
[2] Escola Politécnica – Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav 3, n. 180
05508-900 – São Paulo – SP - Brazil
{liria.sato, calebe.bianchini, marcelo.esperidiao}@poli.usp.br

Abstract. In this paper we present and evaluate Inhambu, a distributed object-oriented system that relies on dynamic monitoring to collect information about the availability of computational resources, providing the necessary support for the execution of data mining applications on clusters of PCs and workstations. We also describe a modified implementation of the data mining tool Weka, which executes the cross validation procedure in parallel with the support of Inhambu. We present preliminary tests, showing that performance gains can be obtained for computationally expensive data mining algorithms, even when running with small datasets.[1]

1 Introduction

Knowledge discovery in databases is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [1]. Data Mining (DM) is a step in this process that centers on the automated learning of new facts and relationships in data, which consists of three basic steps: data preparation, information discovery and analysis of the mining algorithm output. All of these three steps exploit huge amounts of data and are computationally expensive. In this sense, several techniques have been proposed to improve the performance of DM applications, such as parallel processing [2] and implementations based on cluster of workstations [3] and computational grids [9]. These techniques can leverage the deployment of DM applications to production scales.

Building computer clusters for high performance computing has gained increasingly acceptance during the last few years. According to the November 2003's list of the top500 supercomputer sites [4], 208 were clusters, whereas only 93 clusters ap-

[1] This project is partially granted by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico, Brazil), under contract number 401439/2003-8.

H. Jin et al. (Eds.): NPC 2004, LNCS 3222, pp. 213-220, 2004.

peared 12 months before. Assembling computer clusters comprised by commodity PCs or workstations is an easy roadmap to provide computational power at low cost. The effective use of clusters forcibly passes thru the availability of software tools capable to support the efficient usage of their resources. In networks of workstations or PCs, individual computers may present low indices of utilization of computational resources, so that idle resources can be used for the execution of processing and memory intensive applications. In this sense, scheduling policies should involve dynamic detection and allocation of idle resources..

In this paper, we present and evaluate Inhambu, a distributed object-oriented system that provides load monitoring and detection of idle resources in the computers that are part of the cluster. The adopted policies take into account real situations in which computers may be heterogeneous and the availability of resources may fluctuate due to presence or absence of local users. Although Inhambu can be used to provide resource management to other applications, in the context of this project it is used for supporting the execution of the Weka System [5], which is an open source software for data mining.

The remainder of this paper is organized as follows: Section 2 introduces the architecture and policies implemented by Inhambu, whereas Section 3 outlines the changes made to Weka in order to run data mining tasks in a parallel fashion. Section 4 goes on to present performance results, and Section 5 outlines some related work. Finally, Section 6 summarizes our results and outlines future work.

2 Overview of Inhambu

This section briefly describes the main components and strategies comprising our system named Inhambu, whose main components are depicted in Figure 1. Thru this architecture, Inhambu implements an extended *trading service* which can support resource management functionalities to the interaction among client/server programs written in Java. The *trader* enables server programs to publish their services, by invoking its *exportService* operation and passing service names and remote object references. Such information remains stored in the *trader*, which can be queried by client programs by means of its *importService* operation. Clients must import references to remote objects that implement the application services before they can invoke them. Clients can be either application programs, or the Weka's user interface.

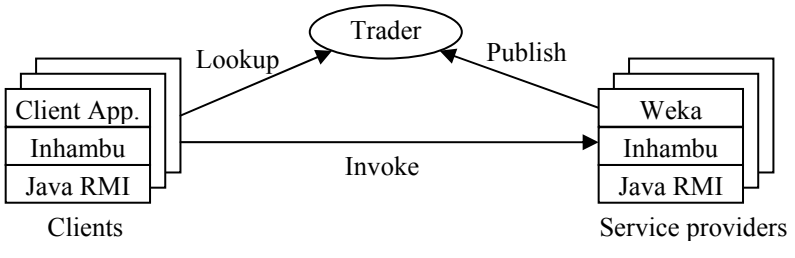

Fig. 1. The Trader Model, implemented by Inhambu.

During the execution of the import operation, a resource management policy that aims at minimizing the execution times is used. Such policy uses information about idle resources, (e.g. CPU, memory) which is periodically received from monitoring agents placed on every server computer. In summary, our policy looks for the *best computer* which implements the requested service. If such server can be found, then an object reference is returned to the requesting client. If no good server currently implements the requested service, the trader looks for another good computer to instantiate a new object that implements the service (which could be the first instance for this service type, or a new replica of existing ones). In all cases, the server selection procedure looks for the computer capable to execute the service in the shortest time, taking into account its processing power and current load. A more detailed description of the system's architecture and operation was presented in [6].

3 Parallelizing Cross Validation with Inhambu

Weka [5] is currently one of the most popular DM tools. It is an open source software developed by researchers at The University of Waikato, New Zealand, and issued under the GNU General Public License (GPL). Weka is written in Java, being currently available to Windows, MAC OS and Linux platforms. In a nutshell, Weka provides implementations of several algorithms for data mining. Its current version contains implementations of 71 algorithms for classification, 5 algorithms for clustering, 2 algorithms for association, and 12 algorithms for attribute selection. In fact, it is continually growing, incorporating more and more data mining algorithms. All these algorithms can be either applied directly to a dataset or called from Java codes.

Classification algorithms, also called classifiers, are predominant in relation to the other ones implemented in Weka. Therefore, we decided to first investigate potential benefits that Inhambu could bring in relation to the classifiers package. In this sense, the standard way of predicting the error rate of a classifier given a simple, fixed sample of data is to use stratified tenfold cross validation [5]. In the tenfold cross validation process, the dataset is divided into ten equal parts (folds) and the classifier is trained in nine parts (folds) and tested in the other one. This procedure is repeated in ten different training sets and the estimated error rate is the average in the test sets. Clearly, these experiments can be performed in a parallel way, taking advantage of Inhambu. It is important to emphasize that all classifiers implemented in Weka can benefit from such approach.

In this paper, we focus on the parallelization of the cross validation by distributing different folds to execute in different nodes of the cluster. To accomplish this, we modified the *classifiers* package. The most important class in this package is the `Classifier`, which defines the general structure of any scheme for classification. All classifier algorithms are implemented as subclasses of *classifier*. It contains two methods, named `buildClassifier()` and `classifyInstance()`, which must be implemented for each classifier method, so that the whole scheme redefines them according to how it builds a classifier and how it classifies data instances. The *classifier* creates a uniform interface for building and using all the classifiers methods, so that the same evaluation module can be used to evaluate the performance and

accuracy of any classifier in Weka. The evaluation module is implemented by the class *Evaluation*, whose method `crossValidateModel()` implements the stratified *n-fold cross validation* procedure, in which the dataset is divided into *n* equal parts. Then, the algorithm is trained in *n-1* parts and tested in the remaining one. This procedure is repeated in *n* different training sets and the estimated error rate is the average in the test sets. Clearly, these experiments are independent and can be performed in parallel by up to *n* computers of the cluster. Although the number of folds in cross validation can be chosen by the user, the tenfold cross validation (i.e., setting n=10) is popularly known as a good practice among data mining practitioners and researchers. If no other value to *n* is provided by the user, Weka assumes ten (by default).

4 Performance Tests

In order to evaluate the performance of Inhambu, we have performed several simulations by means of two classifiers that are popular in the data mining community: PART and Multilayer Perceptrons. In summary, the PART classifier provides rules from pruned partial decision trees [5]. Multilayer perceptrons are feedforward neural networks that learn by means of backpropagation algorithms [10], which are gradient descent techniques with backward error (gradient) propagation. Our simulations were performed in three datasets that are benchmarks for data mining methods: Iris Plants, Wisconsin Breast Cancer, and Congressional Voting Records. These datasets are available at the UCI Machine Learning Repository [7] and describe classification problems. The Iris Plants dataset consists of three classes (Setosa, Versicolour and Virginica), each one formed by 50 examples of plants. Each plant is described by four continuous attributes (sepal and petal length and width). In the Wisconsin dataset, each object has nine ordinal attributes and an associated class label (benign or malignant). The total number of objects is 699 (458 benign and 241 malignant), of which 16 have a single missing feature. We removed those 16 objects and used the remaining ones. The Congressional Voting Records dataset includes votes for each of the U.S. House of Representatives Congressmen on 16 key votes (attributes). There are 435 instances (267 democrats, 168 republicans) and each of the 16 attributes is Boolean valued. However, there are 203 instances with missing values. These instances were removed and we employed the 232 remaining ones in our simulations.

In our simulations, we have employed Weka 3.4.1, the most recent version by the time of this writing. For remote method invocation, Inhambu currently uses the Java/RMI platform, which allows the invocation of either local or remote methods transparently. A test scenario was created that implements a *remote cross validation* service to be executed on a replicated pool of server hosts. In order to enable the cross validation to execute in parallel, we implemented a new class named *ParallelEvaluation*, which inherits the functionalities of the class *Evaluation*.. This new class creates a pool of local threads that implement the preparation of datasets, the invocation of remote cross validation services, and gathering of the results.

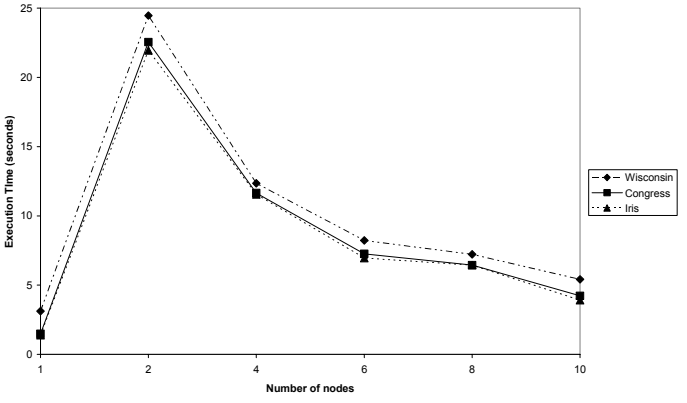

Fig. 2. The execution times for the PART algorithm.

To execute performance tests, we used a cluster composed of 16 PCs interconnected by a 100 Mbps Ethernet switch. Each cluster node is a Celeron 433 MHz single processor with 128KB of cache, 128 MB of memory, running Linux operating system and JDK 1.4.1. Initially, we executed the PART algorithm to analyze the datasets mentioned above in 1, 2, 4, 6, 8 and 10 nodes of the cluster. The *one node* test refers to a sequential execution with the original, non-modified Weka software. Each test was carried out ten times, and the average is illustrated in Figure 2. As one can see, there was no advantage in employing parallel executions to run PART in these datasets. Notice that the execution time with 2 processing nodes is around 8 times slower than local execution. This behavior is typical for PART as well as for other lightweighted classifier algorithms when small benchmark datasets are employed. It is due to the overhead of transmitting one copy of the whole dataset to ten processing nodes (for tenfold cross validation) via RMI. For each invocation, threads are created to manage the transmission process, marshalling and unmarshalling, manipulation of buffers, and so on. The overhead in this case is not paid by the short execution times (which are less than 0.2 or 0.3 seconds for this example). However, it is likely that performance gains can be obtained when applying PART to real-world datasets.

Another set of experiments was carried out in the same environment to analyze the same datasets, but using the algorithm Multilayer Perceptrons instead. The execution times for these experiments are depicted in Figure 3. In this case, the overall execution time can be reduced by a factor of 3 or 4, for a cluster with 10 processing nodes. It is important to emphasize that, in real-world data mining applications, in which huge databases are common, the performance gains are likely to be even more relevant.

In these experiments, all the computers were dedicated to execute our experiments. This measure assures the test is not influenced by other users or applications. Although in real situations the computers are not dedicated and may present fluctuating loads, Inhambu acts by selecting only those nodes which are currently idle, or present good conditions to execute tasks.

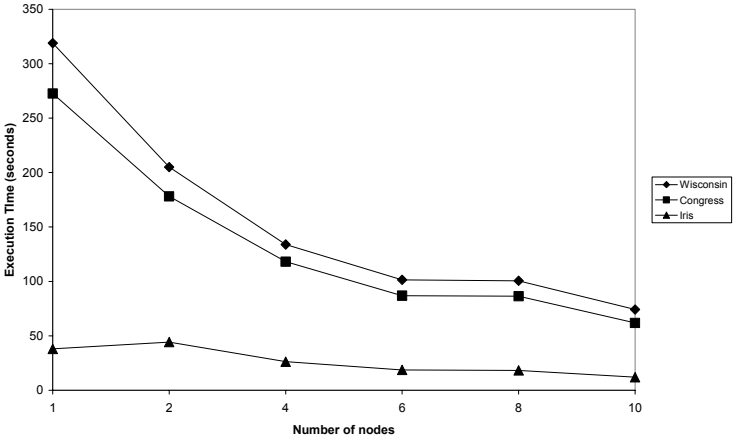

Fig. 3. The execution times for Multilayer Perceptrons.

5 Related Work

Weka-Parallel [8] is also a project that aims at providing a modified, parallel implementation of Weka. Weka-Parallel leverages parallel processing by distributing different folds of the *cross validation* to different computers of a local network. The similarities between Inhambu and Weka-Parallel are evident, since they either focus on parallelizing cross validation, and they operate on local networks of computers. However, some important differences should be highlighted here. Weka-Parallel uses *sockets* as the inter-process communication mechanism, whereas Inhambu uses RMI. The later allows Weka methods to being invoked either locally or remotely with full transparency from location details, and without the need to modify the Weka's classes. This can minimize the handy work to customize new versions of Weka to work on the top of Inhambu. In addition, Inhambu provides some important performance management functionalities, such as the capacity to deal with load fluctuations as well as the heterogeneity of the computers in the cluster. In contrast, Weka-Parallel implements a *round robin* scheduling policy, which does not consider neither the current utilization of the machines nor the differentiated processing capacities. Finally, it is worth to notice that Inhambu aims at supporting the efficient execution of a wide range of high performance applications, not restricted to Data Mining. In the context of this project, however, both Inhambu and Weka are being customized to work together.

6 Conclusion and Future Work

Data mining applications exploit huge amounts of data and are computationally expensive, demanding high quantities of computational resources such as processor

cycles and main memory. Clusters comprised by commodity PCs interconnected by a local network can provide these resources at low cost for data mining practitioners. However, managing the heterogeneity and the fluctuating load of the computers in the cluster are necessary for the effective utilization of clusters of PCs. Inhambu implements mechanisms and policies that rely on dynamic monitoring to collect information about the availability of resources, providing the necessary support for the execution of data mining applications on clusters of PCs and workstations.

In this paper we describe a straightforward scheme to execute cross validation in parallel. Parallelizing cross validation is worthwhile, because it is widely used to evaluate data mining algorithms and it is the most expensive part in this process. In addition, our parallel cross validation scheme can be applied for all the 71 classification algorithms currently implemented by Weka (which implements 90 algorithms). However, deciding to use or not to use parallel processing to execute data mining applications may not be trivial. Due to loose coupling of the architecture, only the execution of sufficiently expensive tasks can be advantageous. Our experiments showed that, depending on the size of the dataset, the parallel execution of PART may not be advantageous, and suggests that it can be a typical behavior for other lightweighted algorithms like this one. However, we believe that parallel execution may be advantageous even for lightweighted algorithms, depending on the characteristics of the dataset, such as the number of instances, number and type of attributes, and the potential patterns that can be found by data mining. Our experiments also showed that more computationally expensive algorithms like the Multilayer Perceptrons always provide performance gains, even with small benchmark datasets. Besides, another common situation in data mining involves evaluating the error rate of several classifiers before choosing the best one, according to a particular application and dataset. In this situation, several cross validations could be performed in parallel, one for each classifier, increasing the degree of parallelism and potential gains.

It is worth to notice that the scheme proposed here, to execute cross validation in parallel over Inhambu, is quite straightforward and obvious. No optimization was used. However, one can notice that in parallel tenfold cross validation, the same dataset is sent up to ten different nodes. Clearly, some scheme to group tasks to execute in the same node of the cluster can reduce the transfer of datasets over the network. Such a scheme may be particularly more advantageous when several cross validation processes are managed to execute in parallel (e.g. to perform the evaluation of several classifiers in parallel). In the near future, we are going to investigate this approach, as well as its application to real-world databases, in which huge amounts of data are processed.

References

1. Fayyad, U. M., Shapiro, G. P., Smyth, P. “From Data Mining to Knowledge Discovery: An Overview”. In: Advances in Knowledge Discovery and Data Mining, Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., Editors, MIT Press, pp. 1-37, 1996.
2. Freitas, A.A., Lavington, S.H., *Mining Very Large Databases with Parallel Processing*, Kluwer Academic Publishers, 1998.

3. Baraglia, R. et al., Implementation Issues in the Design of I/O Intensive Data Mining Applications on Clusters of Workstations, Proc. of 3rd Workshop on High Perf. Data Mining, IPDPS'2000.
4. The TOP500 supercomputer sites list. Available online at: http://www.top500.org
5. Witten, I. H. and Frank, E. *Data Mining: Practical machine learning tools with Java implementations*.. Morgan Kaufmann, San Francisco, 2000.
6. Senger, H., Sato, L. M. Load Distribution for Heterogeneous and Non-Dedicated Clusters Based on Dynamic Monitoring and Differentiated Services. In: Proc. of the IEEE Intl. Conf. on Cluster Computing, pp. 199-206. Hong Kong. Computer Society Press, Los Alamitos, 2003.
7. Merz, C.J., Murphy, P..M., UCI Repository of Machine Learning Databases, http://www.ics.uci.edu, Irvine, CA, University of California.
8. Celis, S., Musicant, D. R. Weka-Parallel: Machine Learning in Parallel. Technical Report. Carleton College Computer Science, 2002. Available at: http://www.mathcs.carleton.edu/weka/report.pdf
9. Canataro, M., Talia, D.The Knowledge Grid,Communications of the ACM, v.46, n.1, 2003.
10. Fu, L., Neural Networks in Computer Intelligence, McGraw Hill, 1994.