DIXER – Distributed Executor for Rough Set Exploration System

Jan G. Bazan[1], Rafał Latkowski[2], and Marcin Szczuka[3]

[1] Institute of Mathematics, University of Rzeszów,
ul. Rejtana 16A, 35-959 Rzeszów, Poland
bazan@univ.rzeszow.pl
[2] Institute of Informatics, Warsaw University,
ul. Banacha 2, 02-097 Warszawa, Poland
R.Latkowski@mimuw.edu.pl
[3] Institute of Mathematics, Warsaw University,
ul. Banacha 2, 02-097 Warszawa, Poland
szczuka@mimuw.edu.pl

Abstract. We present the Distributed Executor for RSES (DIXER) which is a supplementary software for the Rough Set Exploration System (RSES). It takes an advantage of grid computing paradigm and allows to shorten the time necessary for experiments by employing all available workstations to run a scenario of experiments. DIXER software includes most important rough set classification algorithms from RSES and also other algorithms for distributed machine learning. It creates an easy to operate and utilize platform for grid computations and provides a robust and fault tolerant environment for computation-heavy experiments. We provide also experimental evaluation of DIXER that proves at least 96% efficiency in parallelization.

1 Introduction

Machine Learning researchers commonly face a problem of carrying out a huge number of experiments or one, but very time-consuming. The two main reasons of that is the demand for completeness of experimental evaluation and complexity of the machine learning methods themselves.

Let us take, for example, an imaginary algorithm with three parameters. Each of these parameters may take one of three discrete values. This makes $3 \cdot 3 \cdot 3 = 27$ different configurations for our algorithm and if we would like to do a complete empirical evaluation we should provide experimental results for all 27 possibilities. Let's also assume that we have an implementation of this algorithm that requires one minute of execution time to complete single run. To get results of ten experiments using ten-fold cross-validation (CV10) with this algorithm we have to wait $27 \cdot 10 \cdot 10 = 2700$ minutes (45 hours).

The example above is only imaginary, but in real cases we usually face much more demanding problems. If we could acquire four times more computational power then instead of 45 hours we would obtain results overnight. That is why for carrying out experiments we would like to utilize not one, but several computers.

D. Ślęzak et al. (Eds.): RSFDGrC 2005, LNAI 3642, pp. 39–47, 2005.

There are several ways to get more computational power through balancing the load over a number of computational units. The first and the simplest way to do this is just by dividing the work manually and then, by going from machine to machine, put parts of work on all available computers. This approach can be improved by using remote access to another computer but, unfortunately, this method is not available on some software platforms. Obviously, in such an approach we waste a lot of effort on manual partitioning of work, not-automated process of experiment execution, result gathering, and execution monitoring.

Another approach is to use specially designed parallel computers or clusters of computers. However, these machines are extremely expensive and efficient utilization of such resources require re-implementation of software used in experiments (or manual partitioning of work as above).

In late 90's another approach has been proposed, namely the grid computing (cf. [5]). This approach is constantly gaining popularity in academic and commercial circles. In contrast to cluster computing, the grid computing does not require highly efficient and very expensive inter-computer connections and assumes utilization of regular network of workstations, exactly like those available at every university or company. Probably the first widely known grid computing scientific application is the Seti@Home (cf. [10]). In this project every computer connected to the Internet can participate in searching for extra-terrestrial civilization by analyzing data from a radio-telescope.

In this paper we present a grid computing platform dedicated to machine learning experiments. The *Distributed Executor* (*DIXER*) software system is designed to meet the need for a tool that makes it possible to perform series of complex experiments by employing the *grid computing* paradigm. It provides an easy to operate environment that automates execution of tasks in heterogeneous, commonly available networks of workstations. The DIXER is a supplementary software for the *Rough Set Exploration System* (RSES, see [1,2]). It is mainly suited for experiments based on RSES-Lib software [1], but as an open architecture it facilitates a plug-in mechanism, so that new modules can be added without modification of the main program.

The DIXER had to be written from scratch, as existing solutions proved to be not suitable for our purposes. We considered utilization of existing clustering/concurrency support systems, such as *PVM* or *MPI*, but they are either not efficient enough in machine learning applications or provide too low-level solutions. It is not a surprise, as these approaches are designed to work with specialized parallel computers or very expensive equipment like Myrinet network in the case of Network of Workstations (NOW) project (cf. [3]). In the case of grid computing applications in data analysis we should avoid implementations that assume too frequent synchronization between concurrent processes. In general, we would like to make the use of the machine standing in the office next door after hours, or to employ a student lab as a grid machine, when there are no classes.

In the second section we provide a very brief description of DIXER. Next, (Section 3) we describe two modules bundled with DIXER. In Section 4 we describe our experimental evaluation of the efficiency boost achieved through the use of DIXER. Finally, conclusions and bibliography close the paper.

2 Description

The DIXER software consist of two applications: Manager and Node. As they are written in Java, it is possible to install them on variety of operating systems including Linux, MS-Windows, Mac-OS, and various flavors of Unix. *DIXER-Node* has to be installed and run on all computers that cooperate in distributed computation. *DIXER-Manager* should run on one machine only (see Fig. 1). DIXER-Manager schedules experiments (jobs) for all nodes in the grid of computers, i.e., those, where DIXER-Node is executed. DIXER-Node performs all experiments (jobs) scheduled by DIXER-Manager.

The DIXER software creates an easy to operate and utilize platform for grid computing and provides a robust and fault tolerant environment for computation-heavy experiments. At the moment it is bundled with two modules for distributed computations. The *RSES Parallel Experimenter* (RSParallel) allows performing experiments with classifiers available in RSES-Lib. The *Branch-and-Bound* DIXER module makes it possible to perform feature selection with use of wrapper attribute evaluation and strategies similar to well known branch-and-bound search algorithm.

The DIXER-Manager has a graphical user interface (GUI, see Fig. 2) that provides the console for managing all cooperating computers. Playing the boss, for most of the time DIXER-Manager consumes only a small fraction of system resources, both CPU time and system memory. This load, however, depends on the number of nodes in grid and the task at hand. DIXER-Manager requires some additional data files to be created. These files are necessary to store experiment control data.

The DIXER-Node contains simplistic graphical user interface (GUI, see Fig. 2) on which one can supervise the program status. On demand this GUI can be disabled and then DIXER-Node runs silently in background. Usually, DIXER-Node consumes a lion share of system resources, both CPU time and system memory. DIXER-Node can also require some disc space in order to store data files needed in experiments, which are sent by DIXER-Manager together with job descriptions.

DIXER also provides mechanisms for recovery from failure. Execution of all experiments can be easily stopped and restarted without loss of any already computed results. This property is used also for assuring robustness. Should one or more nodes crash because of the hardware or software failure, the already

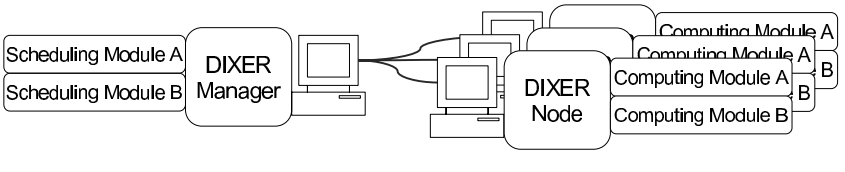

Fig. 1. Architecture of the grid of computers running DIXER-Manager and DIXER-Nodes

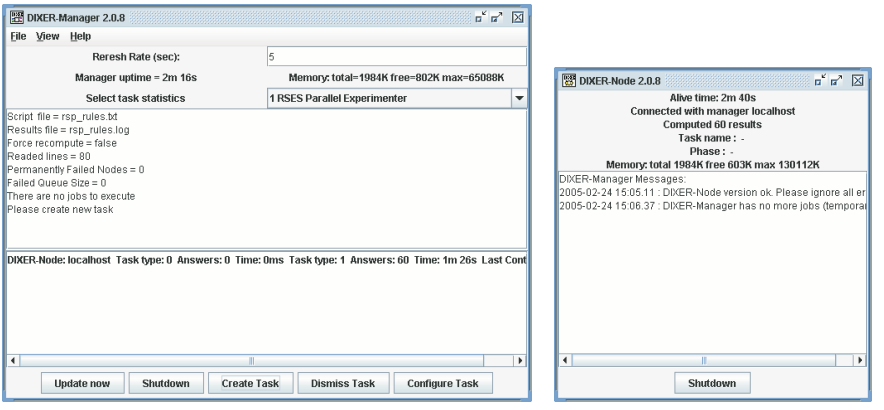

Fig. 2. Graphical User Interface of DIXER-Manager and DIXER-Node

computed results are preserved. If crash occurs only on the computers that run DIXER-Node, then the whole system is still working, but with less computational power due to reduced grid size. If crash touches the computer that runs DIXER-Manager then the DIXER-Manager should be restarted from the point of last delivered result.

There are several advantages of using DIXER for machine learning experiments. The obvious benefit from DIXER is the automation of processing multiple experiments using several computers. If it is necessary to test, e.g., which discretization or which decision rule shortening factor gives the best classification for particular data set, then DIXER can help in automation of experiment series. Another advantage is that DIXER architecture offers the platform that efficiently utilize power of available computers. DIXER is entirely written in Java™ programming language, what makes possible the construction of the grid in heterogenous hardware and software environments. DIXER, as well as the RSES is distributed free of charge for non-commercial purposes. It can be downloaded from the RSES Web page [9].

3 Implemented Methods

The DIXER software is distributed with two built-in modules. Apart from them it is possible to plug-in new modules that implement other distributed algorithms. The already implemented methods are: feature selection and execution of multiple experiments.

3.1 Multiple Experiments

DIXER allows to distribute multiple experiments across all computers that run DIXER-Node. The RSES Parallel Experimenter (RSParallel) is a DIXER module that makes it possible to carry out experiments based on methods implemented in Rough Set Exploration System (RSES), and particularly in RSES-Lib (c.f.

[1]). It processes a special script language, described in DIXER User Manual [7], that is used to prepare individual scenarios for experiments. This script language is interpreted line by line from the beginning of the file to its end, and each line contains description of a separate experiment.

While preparing the scenario file user can choose between several classification algorithms implemented in RSES-Lib. The description of each experiment includes the selection of algorithm, its parameters, and reference to data files used in training and testing phases. The results of experiments are written to a special file that can be used for generating description of results in the form of tables and/or statistics. These results are easy to analyze and support reporting.

Currently RSParallel provides the following RSES-Lib methods in experiment scenarios: discretization, decision rule induction, decision rule shortening and decomposition-based classifiers. A given experiment scenario can be stopped and restarted without loss of already computed partial results. User can also force DIXER to recalculate all results from the beginning.

3.2 Feature Selection

The feature selection problem is one of the most important problems in Data Mining as well as one of the most computationally expensive. The *Branch-and-Bound* DIXER module allows to carry out experiments with very expensive and accurate feature selection based on the wrapper method (see, e.g., [4]), which is a non-rough set feature selection approach. The strategy for searching the space of the possible attribute subsets is similar to the branch-and-bound search. Here, however, this strategy is not complete and is used rather in selection of the first candidate to visit within the lattice of the possible feature subsets. The results are saved in the special file that can be viewed both manually and with a help of text processing tools. As a wrapper classifier the decision tree from RSES-Lib is used. This decision tree algorithm can perform clustering of symbolic values using the indiscernibility as a measure for heuristic split.

This algorithm is an example of method that cannot be manually partitioned in order to distribute work among several machines. It cannot be prepared beforehand as the search strategy is modified on the basis of partial results. The integrated solutions for distributed computing give the advantage over manual methods, because implementation of more sophisticated experiments becomes viable.

The details on using these two methods are provided in the DIXER User Manual [7] bundled with the software.

4 Experimental Illustration

DIXER is available for download since 2003 (see [9]) and already has been used in several research experiments. As an example of such experiment we refer the reader to [8] where research on flexible indiscernibility relations and, particularly, on attribute limited indiscernibility relations is presented. The attribute limited indiscernibility relations are modifications of the indiscernibility relations that

enable different treatment of missing attribute values for each attribute. The experimental work consisted of discretization and induction of decision rules over train part of data using all possible attribute limited indiscernibility relations and calculating classification accuracy over test part of data. Since the RSES-Lib does not support decision rule induction using attribute limited indiscernibility relations a special data preprocessing routine was used to simulate it. Generally speaking, there were two possible treatments of missing values for each attribute, so the number of all possible attribute limited indiscernibility relations is 2^N, where N is the number of conditional attributes in data. The experiments were carried out for five data sets, three of them with six conditional attributes and two of them with eight conditional attributes. Every data set was partitioned into ten pairs of train and test samples in order to use ten fold cross-validation (CV10), so in total there were $10 \cdot (3 \cdot 2^6 + 2 \cdot 2^8) = 7040$ experiments. The execution of those experiments took about a week on heterogeneous grid of computers consisting of up to three computers with Pentium 4 2.6GHz CPU and one computer with Pentium M 1GHz CPU, connected with 100Mb switched network. Some of computers were detached from the grid because of network failure or other user actions, but as it was mentioned before, it did not break the computation, it only reduced the computational power of the grid.

For the purpose of this paper we repeated a part of the experiments described above on a homogeneous grid of computers. The experiments were carried out with use of up to ten identical computers with a 733Mhz Pentium III CPU connected to 100Mb non-switched network. We took one pair of train and test data that has the shortest time of discertization and decision rule induction. The selected data — head injury data (hin), 3 classes, 6 categorical attributes, 1000 observations, 40.5% incomplete cases, 9.8% missing values — requires less than 7 seconds on a computer mentioned above to complete a single decision rule induction procedure. The number of possible attribute limited indiscernibility relations, thus the number of experiments is $2^6 = 64$. We selected this set of data because with so small execution time the effect of communication and synchronization overhead will be most visible. We may say, that this experiment is the worst case as the potential communication overhead is significant.

The results of experiments are presented in Table 1. The experiments were carried out using separate machine for manager and separate machines for nodes of the grid. We decided to measure the execution time in that way, because the execution of all 64 experiments on exactly the same computer acting as manager and node requires up to two seconds more than the execution on separate computers. In the first row of Table 1 the number of computers acting as nodes is presented, in the second — the execution time in seconds, and in the third row — the relative speed-up factor.

We have to take into account that on some configurations, where 64 is not divisible by the number of nodes in grid, DIXER-Manager has to wait for the last computer to collect all results. This implies that on configurations where 64 is divisible by the number of computers cooperating in grid the efficiency of grid should be slightly better. The worst result, i.e., result with largest difference

Table 1. Execution time and observed speed-up of DIXER grid consisting of up to ten computers

Number of computers	1	2	3	4	5	6	7	8	9	10
Time	440.32	220.17	148.28	115.13	92.81	75.52	65.72	55.79	50.87	46.3
Speed-up	1.00	2.00	2.97	3.82	4.74	5.83	6.70	7.89	8.66	9.51

from the expected speed-up is the result with ten computers acting as grid nodes. In this case we have 9.51 speed-up ratio instead of 10. That corresponds to two wasted seconds. In order to measure the scalability more accurately, we also performed a linear regression on the presented results. The results are linearly approximated by line with factor 0.9636 with goodness of fit $R^2 = 0.999$. It corresponds to 96% efficiency in computational resources utilization.

Figure 3 presents two charts with results of experiments. On that figure the execution time and speed-up for first 16, 32 and 48 experiments is presented for reference purposes. On the execution time chart (left) we can notice that the reduction of computation time is inversely proportional to the resources (number of computers). On the right, speed-up chart a reference line with ideal speed-up is presented for comparison. We can observe that the lowest speed-up curve correspond to the execution of first 16 experiments. It suggests that the DIXER achieves worst parallelization efficiency is at the beginning of distributed computation. However, overall efficiency is never lower than 95% of the ideal speed-up.

The excellent results of experiments are partially caused by the fact, that on mentioned computers we carried out multiple experiments utilizing always the same 64 pairs of data. This means, that on those computers we needed to download the input data to local hard disk only once. However, if the data for scheduled experiment is not present on the remote computer then they have to be sent from the Manager to Node using DIXER internal mechanism for data

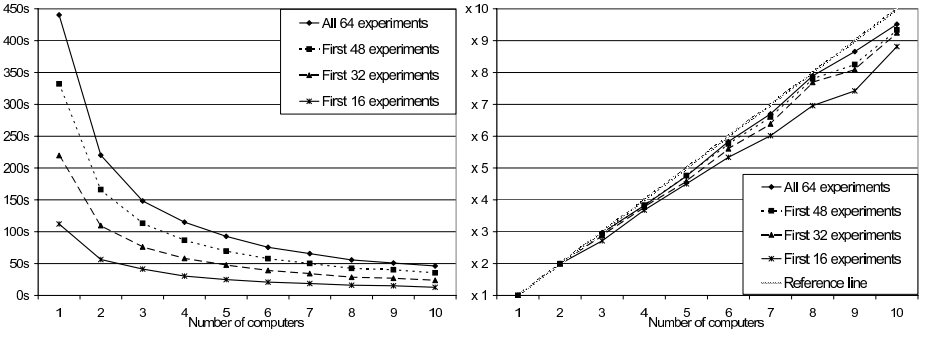

Fig. 3. The left chart presents execution time of the experiments on DIXER grid with parition to first 16, 32, 48 and all 64 experiments. The right chart presents observed speed-up of the execution.

transfer. We have not tested in detail the influence of data transfer on execution time, but this influence should not be significant. There are two arguments supporting such statement. Firstly, the size of the data is somehow proportional to the execution time of the experiment. In this particular case each pair of files have total size of about 19KB, so the transfer time is negligible. Secondly, even when the data is on local hard disk, DIXER performs additional test by comparing properties of these files on the DIXER-Manager computer with originals stored on the DIXER-Node computer. It checks such values as: last modification time, total size and CRC16 of the first 4KB of the file. Such a verification is very similar to the data transfer, as it employs both local hard drive and network connection. This verification is, of course, included in the results presented in Table 1 and Figure 3. After all, not all experiments done with help of DIXER utilize different data files. More typical use is to test different methods or different parameters on relatively small set of data files.

5 Conclusions

The DIXER software provides a platform for grid-like computations supporting Machine Learning and Data Mining experiments. The computational core, with respect to the Machine Learning algorithms, is based on renewed RSES-Lib library that implements several Rough Set based algorithms. The flexibility of DIXER software makes it possible to extend its functionality without modification of the essential source code. The DIXER is easy to operate and provides a robust and fault tolerant environment for computational-heavy experiments. Especially in an academic environment, where usually a number of standard workstations is available, it is the cheapest way to access an enlarged computational power. The minimization of communication overhead and grid management (load balancing) makes it possible to obtain the computational throughput that almost equals the sum of computational powers of cooperating computers.

Acknowledgments

We wish to thank supervisor of RSES and DIXER projects, professor Andrzej Skowron, and rest of the team participating in design and development: Nguyen Hung Son, Nguyen Sinh Hoa, Michał Mikołajczyk, Dominik Ślęzak, Piotr Synak, Arkadiusz Wojna, Marcin Wojnarski, and Jakub Wróblewski. The research has been supported by the grant 3T11C00226 from Ministry of Scientific Research and Information Technology of the Republic of Poland.

References

1. Bazan, J.G., Szczuka, M.S., Wróblewski, J.: A new version of rough set exploration system. In Alpigini, J.J., Peters, J.F., Skowron, A., Zhong, N., eds.: Rough Sets and Current Trends in Computing, Third International Conference, RSCTC 2002, Malvern, PA, USA, October 14-16, 2002, Proceedings. Volume 2475 of Lecture Notes in Computer Science., Springer (2002) 397–404

2. Bazan, J.G., Szczuka, M., Wojna, A., Wojnarski, M.: On the evolution of rough set exploration system. In Tsumoto, S., Słowiński, R., Komorowski, H.J., Grzymała-Busse, J.W., eds.: Rough Sets and Current Trends in Computing, RSCTC 2004. Volume 3066 of Lecture Notes in Computer Science., Springer (2004) 592–601
3. Culler, D., Arpaci-Dusseau, A., Arpaci-Dusseau, R., Chun, B., Lumetta, S., Mainwaring, A., Martin, R., Yoshikawa, C., Wong, F.: Parallel computing on the Berkeley NOW. In: Proceedings of the 9th Joint Symposium on Parallel Processing, JSPP'97. (1997)
4. Dash, M., Liu, H.: Feature selection for classification. Intelligent Data Analysis **1** (1997) 131–156
5. Forester, I., Kesselman, C. (Eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufmann. (1998)
6. Groupp, W., Lusk, E.: Why are PVM and MPI so different? Report of Mathematics and Computer Science Division of Aragonne National Laboratory ANL/MCS-P667-0697. (1997)
7. Latkowski, R.: DIXER User Manual. Warsaw University. (2003) (refer to [9])
8. Latkowski, R.: Flexible Indiscernibility Relations for Missing Attribute Values. Fundamenta Informaticae **66:2** (2005), In print.
9. The RSES WWW homepage at `http://logic.mimuw.edu.pl/~rses`
10. SETI@home WWW homepage at `http://setiathome.ssl.berkeley.edu/`