Distributed Generative Data Mining

Ruy Ramos and Rui Camacho

LIACC, Rua de Ceuta 118 - 6^o 4050-190 Porto, Portugal,
FEUP, Rua Dr Roberto Frias, 4200-465 Porto, Portugal

Abstract. A process of Knowledge Discovery in Databases (KDD) involving large amounts of data requires a considerable amount of computational power. The process may be done on a dedicated and expensive machinery or, for some tasks, one can use distributed computing techniques on a network of affordable machines. In either approach it is usual the user to specify the *workflow* of the sub-tasks composing the whole KDD process before execution starts.
In this paper we propose a technique that we call *Distributed Generative Data Mining*. The *generative* feature of the technique is due to its capability of generating new sub-tasks of the Data Mining analysis process at execution time. The *workflow* of sub-tasks of the DM is, therefore, dynamic.
To deploy the proposed technique we extended the Distributed Data Mining system HARVARD and adapted an Inductive Logic Programming system (IndLog) used in a Relational Data Ming task.
As a proof-of-concept, the extended system was used to analyse an artificial dataset of a credit scoring problem with eighty million records.

keywords: Data Mining, Parallel and Distributed Computing, Inductive Logic Programming.

1 Introduction

As a result of more complex and efficient data acquisition tools and processes there is, in almost all, organisations huge amounts of data stored. Large amounts of money are invested in designing efficient data warehouses to store the collected data. This is happening not only in Science but mainly in industry. Analysis of such amounts of data has to be done using (semi-)automatic data analysis tolls. Existing OLAP techniques are adequate for relatively simple analysis but completely inadequate for in-depth analysis of data. The discipline of Knowledge Discovery is Databases (KDD) is a valuable set of techniques to extract valuable information from large amounts of data (data ware houses). However, KDD ([1]) is facing nowadays two major problems. The amounts of data are so large that it is impractical (or too costly to download the data into a single machine to analyse it. Also, due to the amounts of data or to its distributed nature in large corporations, it is the case that the data is spread across several physically separated data bases. These two problems prompted for a new area of research called

Distributed and Parallel Data Mining[2]. This new area addresses the problem of analysing distributed databases and/or making the analysis in a distributed computing setting.

There are parallel versions of Decision Trees algorithms [3], parallel Association Rules Algorithms [4] and parallel Inductive Logic Programming (ILP) algorithms ([5],[6]) that may be used in the (Relational) Data Mining step of KDD. To handle the large amounts of data these algorithms these use a data partition approach and distribute the analysis work over a cluster of machines. One weakness of these systems is that if one of the machines fails the whole process has to be restarted, which is a serious drawback for long task's execution time. The implementation of such parallel DM algorithms are not designed with fault-tolerant features.

In this paper we propose a solution to make parallel DM algorithms such as Decision Trees, Association Rules or Inductive Logic Programming to be fault-tolerant and able to run on conventional PCs without disturbing the normal workings of an organisation. For that purpose we have extended the HARVARD [7] system to accommodate such desirable features for the parallel DM algorithms. The HARVARD system (**HARV**esting **A**rchitecture of idle machines fo**R** **D**ata mining) has been developed as a computational distributed system capable of extracting knowledge from (very) large amounts of data using techniques of Data Mining (DM) namely Machine Learning (ML) algorithms. We take advantage of its following features. In a Condor [20] fashion, the system only assigns tasks to idle resources in the organisation. The system may access data distributed among several physically separated databases. The system is *independent* of the data analysis (ML) tool. It has very good facilities to recover from both slave and master nodes failure.

To integrate a parallel algorithm such as parallel Decision Trees, parallel Association Rules or parallel ILP we included in the HARVARD system the possibility of creating new tasks at run time at the HARVARD's task description level. This is the new *generative* feature of the distributed computing characteristic of HARVARD. The parallel DM algorithm on the other hand has to be adapted to be able to suggest new tasks in the HARVARD task description language. We present in this paper an example for the Inductive Logic Programming IndLog [9].

With this proposal we may include fault-tolerant features in some of the most popular distributed DM algorithms.

The rest of the paper is organised as follows. In the Section 2 we present the HARVARD system. In Section 3 we present basic notions of Inductive Logic Programming enough for the reader to understand the coupling of the analysis tool with the HARVARD system. We explain the generative technique for Data Mining in Section 4. The deployment of the HARVARD system extension is described in Section 5. Section 6 compares other projects with features close to HARVARD capabilities. We conclude in Section 7.

2 The HARVARD system

The Architecture

The KDD process is composed of a set of tasks that are executed according to a workflow plan. Both the tasks description and the workflow plan are provided by the user in two files. One file contains the tasks description and the second one contains the workflow. The workflow is specified using a control description language. Each task specification is made using XML. The information concerning each individual task include the name and location of the tool used in the task, the location of the data being processed in the task and the computational resources required (platform type, memory and disc requirements).

The distributed architecture of the HARVARD system is composed of a Master node and a set of computing nodes called Client (or Slave) nodes. The Master node is responsible for the control and scheduling the sub-tasks of the whole KDD process. Each Slave node executes application (sub-)tasks assigned by the Master node. Each node is composed by four modules that execute specific tasks to make the overall system working.

In what follows we refer to Figure **??** for the modular structure of both the Master and the Slave nodes. We now describe in detail the each node type.

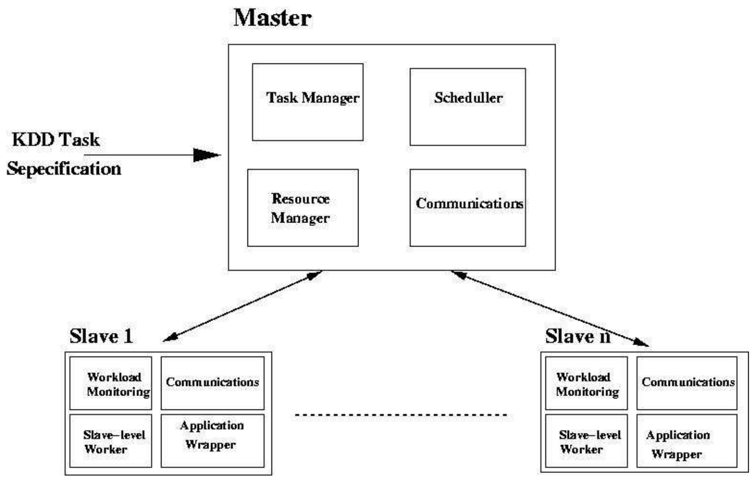

Fig. 1. The Harvard basic system architecture.

The Master node The Master node is responsible for reading the KDD process specification and executing it. Each task is of the KDD process is handle by the system as a **Working Unit** (**WU**). Each WU will be assigned to a one or more machines. The assignment of a WU to more than one machine makes the the system more tolerant to faults. It occurs when there are idle machines available and the task is expected to have long running times. There are other fault tolerant features that we will refer bellow. When a WU finishes the results is associated with that WU and the status of the workflow graph updated. When the graph is completely traversed, meaning that the KDD process has finished, the result is returned to the user.

The Master node is composed by four modules: the Task manager; the Scheduler; the Resource Manager and; the Communications module.

The Task Manager Module The basic function of the **Task Manager** (TM) module is to store and maintain and provide information concerning the tasks of the KDD process. The TM module constructs a graph structure representing the workflow of tasks.

It first reads and stores the specifications of all tasks composing the KDD process and then reads the workflow plan of the tasks and constructs a workflow graph structure. This module updates the status of the tasks in the graph and associates the results of each one when finished. At the end informs the user of the results of the KDD process. It may also be used to monitor the whole KDD process providing the user with information about the task finished, running and waiting computational resources to run.

The TM interacts with the Scheduler module. Looking at the workflow graph this module informs the Scheduler of ready to process tasks, provides a complete specification of each task and receives information concerning the terminations and results of each task.

The Resources Manager Module The **Resources Manager (RM)** module stores and periodically updates information concerning the computational resources usable by the HARVARD system. When the system starts this module loads from a database the static information concerning all the computational resources usable by the system. That information is dynamically updated during the system execution. The information of each resource includes the type of platform and CPU, the amount of memory and disc space and a time-table with the periods the machine may be used. The workload of each machine is communicated periodically to this module to update so the system has a updated view of the resources. The RM module has a method (a match maker) to compute the "best" computational resource for a given request from the Scheduler. Each computation resources has a time-table of availability of the resource and the policy of use. This information state when the machine is available and in what conditions. The usage conditions may indicate that the system may use the machine only when there are no users logged in or by specifying a workload threshold that must be respected at all times.

The Task Manager module receives, from the Scheduler, requests for available machines satisfying a set of resources requirements and returns a best match at the moment. This module alerts the Scheduler that a task must be reassigned in two situations: if a machine is severely delayed to notify the TM module of its workload and; if the pre-established period of use of the machine is expired[1]. The TM module receives periodically the workload of all running machines.

The Communications Module The **Communications (COM)** module is the only channel to access the world outside a node. All messages or requests concerning components or resources outside the node are processed by the COM module. This module exists in both Master and Slave nodes. To accomplish that task it implements several communication protocols that includes: RMI, socket, HTTP and JDBC. All these allows a client to download the task required software (HTTP), download the data (JDBC), send messages to the Master (sockets or RMI) and allows the Master to send messages to the Slaves (socket or RMI). It also allows the Master to keep a DB backup of its status and activities (JDBC) to allow a full recover in case of fault.

This Master COM module interacts via RMI or sockets with the COM module of the Slave to send messages. In a Master node the messages to be sent are received from the Scheduler module or the Resources Manager module. The former sends messages concerning task assignments and control directives whereas the later sends tasks status updated to be stored in a DB (fault tolerant purposes). The COM module receives and redirects the workload messages for the RM module. Received messages concerning tasks results are redirected to the Scheduler module.

The Scheduler Module The **Scheduler** module controls the execution of the tasks composing the KDD process, launching, rescheduling or stopping the Work Units. The scheduler may also decide to assign a WU[2] to more than one Slave node. The scheduler inspects the workflow graph where the tasks interconnections and status are represented to decide what tasks to activate and when.

The Scheduler asks the Resource Manager module for the best match machine satisfying a given Work Unit requirements. With the results of such request the Scheduler assigns that WU to the given Slave and notifies the Slave via the Communications module. Whenever there is a change in the status of a WU the Scheduler is informed by the Task Manager of that event and triggers the (re)scheduling a new task.

[1] In this case the task running on the machine is terminated.

[2] The ones considered more critical for some reason like training longer execution times.

A Slave node A Slave node does the actual data analysis work by running the Data Ming tool. In order to have a distributed system that is independent of the Data Mining tool the DM tool is involved in a wrapper that directly controls the DM tool. Each Slave also reports periodically its workload to the Resource Manager module of the Master. It is through the Slave's Communications module that the Slave downloads the DM tool and the data to be processed, and stores the results of the local analysis.

Each Slave has four modules: the Workload Monitoring (WM); the Worker (SW); the Application Wrapper (AW) and; the Communications (COM) module.

The Worker module The WU message is interpreted in this module. A WU usually results in several steps to be performed. A typical WU for analysing data involves the downloading of the analysis tool, the download of the data, the processing and the return of the results. The Worker module controls all these steps by asking the Communications module to access the software and data and triggering the Application Wrapper module to execute the analysis. Finally it sends (via Communications module) the results to the Master.

The Worker nodes interacts with the Communication modules by sending it request to download the software and data and to return the final result. It also asks the Application Wrapper to start the analysis task and collects the results from it.

The Application Wrapper module The AW module completely controls the DM tool. It supplies the DM tool input stream and collects whatever appears at the DM output stream. Through the input stream the module provides the commands for the DM tool. The commands are provided in a file specified in the Working Unit specification. The output stream is stored in a file as the results file. The results file is uploaded to a database entry as specified in the WU specification. For the time being all the analysis of the results files are done in other follow up WU where special scripts written by the user do the necessary analysis. This permits the system to be independent of the DM tool.

The Workload Monitoring module This module monitors periodically the workload of the machine it is running and reports that information to the Resources Manager module of the Master. It also detects in a user has login into the machine. In that later case the Master is informed that the task running will be terminated. The Slaves enters a idle state where it just waits for the machine to be idle again.

Communications Module The slave Communicating module is the only channel to the outside world. It has capabilities to download software using HTTP or ftp protocol, it may download data from a DB using JDBC and it can send and receive messages to and from the Master using RMI or sockets.

The Communications module interacts with all modules of the Slave node delivering and receiving messages.

Sub-tasks workflow description language

The HARVARD system accepts as input a file describing the workflow of the sub-tasks composing the KDD process. The workflow is a graph with two kinds of nodes: sequential nodes and; parallel nodes. Each node stores a set of tasks to be executed or edges to other nodes. In a sequential node the set has an order and that order represents the sequential execution of the sub-tasks that must be respected. In a parallel node the tasks in the set may start all at the same time. In a parallel node there may be a *barrier* specifying the tasks that must terminate before the "execution" of the node is considered terminated. The graph has a root node where the tasks of the KDD process start executing.

Some of the steps in a KDD process are done quite frequent and most often are the same for a vast number of domains. For example feature subset selection or the DM tool parameter tuning are quite frequent pre-processing tasks in the KDD process. For these frequent tasks the task description language provides a set of macros for these complex operations. For example: to do a top-down feature selection up to two attributes or tune the "p" parameter using the values p1, p2 and p3. The system will then "unfold" those macros into the corresponding graph structure.

3 ILP in a *tiny nutshell*

Inductive Logic Programming (ILP)[10,11] is a discipline in the intersection of Machine Learning and Logic Programming. ILP studies techniques to (automatically) induce models for data. In ILP both the given data and the induced models are represented using First Order Logic. The data provided to an ILP system are of two kinds: i) examples and; ii) background knowledge. Examples are instances of the concept whose definition the system will induce. The background knowledge is any information the user thinks is relevant to construct the model. In the most popular ILP systems the induction process is mapped into a search on the space of all possible hypotheses (the hypothesis space). The system then uses any if well-know search algorithms to search the hypothesis space and return *the best* hypothesis according some specified criterion.

Some advantages of using ILP in DM tasks are the following. It has a very powerful description language to encode the constructed models. The user may give the system easily any information he considers relevant to produce the model. The background knowledge may include not only relation's definitions but also numerical computations like regression models, geometric or statistical models etc. that are nicely combined in the final model. Most often ILP induced models are comprehensible to the user.

A major shortcoming of ILP systems is its efficiency. One possible approach the overcome ILP's lack of efficiency is through the use of parallelism.

As described in [6] there are several approaches to parallelise an ILP system. One of the most simple but providing good results (see [5] for a comparison of several methods) consists in establishing a partition of the data and apply an ILP system to each subset of the data. The models induced by each ILP system on each subset are sent to a master node that combines the models. If the assembled model "explains" all the examples then the process stops and the model is the final model. Otherwise there are a next round of the same procedure where the examples not "explained" are subject a learning process equal to the first step. We have implemented in IndLog this approach to parallel ILP. It is a similar approach to the one described in [2] where classifiers are constructed for each set of the data partition and then sent to a central node where they are combined. In the ILP case this process is repeated until all examples are "explained".

4 Distributed Generative Data Mining

The main cycle of a DM algorithm, such as Decision Trees, Association Rules or ILP is run a number of times that depend, among other things, on the input data. The number of nodes in a Decision Tree is not fixed before the algorithm hat constructs the tree is run. The number of clauses in that and ILP algorithm induces is also not fixed before the algorithm processes the data. When developing parallel versions of these algorithm the number of tasks on each run will depend on the dataset being processed. On the other hand in a KDD process the number of tasks and their workflow is established by the user, using a tool like YALE[12] for example, and stays unchanged during the execution of the KDD process. To be able to integrate a parallel version of a DM algorithm in a automatic KDD setting one as to allow the tool to have a dynamic workflow of the tasks. For example new tasks are generated whenever a parallel Decision Tree constructed splits a new node. To accommodate such situation we extended the HARVARD system to be able to dynamically accept new tasks during run time. New tasks are encoded as new nodes in the graph that represents the task's workflow.

In the data parallel version of IndLog a master node decides the split of the examples and assigns each slave node a subset. Instead of assigning directly the "sub-tasks" to the slaves via MPI interface, the IndLog translates the sub-tasks requests to a format that the Master node (Task Manager module) of the HARVARD understands. The HARVARD scheduler then assigns each task to existing idle machines. If there are too many idle machines then the same task my be run (redundantly) in more than one machine. In case of failure of one of the machine the analysis completes without delay.

The main advantages of the generative technique proposed in this paper is to increase the fault-tolerance of the analysis tool and to take full advantages of the HARVARD system. Using the HARVARD system the analysis does not disturb the normal workings of the organisation, does not require dedicated machines and has tolerance to failure of both the Master node and any of the Slave nodes.

5 Deployment of the HARVARD system

Just to test the feasibility of our approach and not to compare the systems performance on a specific problem we produced a large artificial data set with realistic information. The data set is on the domain of credit scoring. We characterised each instance with 55 attributes that correspond to the 55 fields of an actual form used by a real bank. The information used to generate the records was based on census information publicly available at the Brazilian national statistics office. We used a bank expert to provide the rules that assign the class value to the generated registers. The data set has 80 million registers. The data was stored in three MySQL databases in different machines. We used a laboratory with 15 PCs where Master students have classes and use for developing their practical works. The machines have dual boot so sometimes they start with Linux and sometimes with Windows. It takes several hours to analyse the data set on the reported computational environment using the HARVARD system with IndLog.

To analyse the data using IndLog [13,9] we had to provide scripts for the Application Wrapper of the Client node (see Figure 1) to control the IndLog system. The IndLog system was run in a data parallel fashion allowing a maximum of 50000 examples at each node.

To evaluate the fault-tolerant features we deliberately generated failures in some machines during the analysis process and under two circumstances. First we disconnect a machine running a task that was also assigned to other machine and notice no increase in the analysis time. Secondly we disconnected a machine were a task was running that was not assigned to any other machine. With a small overhead that task was reassigned to another machine an the analysis continued then normally.

6 Related work

Our system is designed to take advantage of idle computers in an organisation and adequate for problems that may be decomposed into coarse grain sub-tasks. We present some related projects that can reach this objective but of differentiated form our architecture.

The Globus Alliance [14] is an international collaboration that does research in grid computing that seeks to enable "the construction of computational grids providing pervasive, dependable, and consistent access to high-performance computational resources, despite geographical distribution of both resources and users". One of the results of this research is "The Globus Toolkit"that is a "bag of services"like: resource allocation manager that provides creation, monitoring and management services; security infrastructure; monitoring and discovery services. For each of their services there is a programming interface in programming language C.These core services are used by other organisations and Globus to make high level components and systems [15,14,16].

The Boinc (Berkeley Open Infrastructure for Network Computing)[3] is a platform that makes it easy for scientists to create and operate public-resource computing projects. Workstations connected to the Internet by phone or DSL line can be participate some project and share its own power computer to solve scientific problem when device is idle. The process is very sample, people interested to participate just installing a software client that connect a project master server. So, when workstation is idle some tasks may be executing. Some projects like SETI@home, Folding@home using the Boinc platform [17].

The Knowledge Grid is a specialised architecture in data mining tools that uses basic global services from Globus architecture [18]. The architecture design for Knowledge Grid following some principles: data heterogeneity and large data sets handling; algorithm integration and independence; compatibility with grid infrastructure and grid awareness; openness, scalability, security and data privacy [19].

Condor operates in workstation environment. The system aims to maximise the utilisation of workstation with as little interference as possible between the jobs is schedules and the activities of the people who own workstations. "Condor is specialised job and resource management system for compute intensive jobs. Like other full-featured systems, Condor provides a job management mechanism, scheduling policy, priority scheme, resource monitoring and resource management. Users submit theirs jobs to Condor when and where to run the based upon policy, monitors their progress, and ultimately informs the user upon completion"[20]. Condor allows almost any application that can run without user interaction to be managed. This is different from systems like Set@home and Protein Folding@home. These programs are custom written. Source code does not have to be modified in anyway to take advantage of these benefits. Code that can be re-linked with the Condor libraries gain two further abilities: the jobs can produce check-points and they can perform remote system calls [20].

Like the project Boinc, our architecture intends to use of idle workstations and also the system considers the heterogeneous environment with different operational systems (Linux, Windows, OS-X) but instead of it has a light client installed in each workstation it uses the Java Virtual Machine. A new approach will be present that to use old applications for data mining without re-build or re-compile with a new libraries like Condor or another approaches.

Besides, our proposal implements two-level language. A specific semantics for the administration of the data mining process, and other for specification of tasks of distributed processing. While a language is destined to the user for the definition of the process of the knowledge discovery, the other language is used by the system to manage the distributed processing.

[3] **University of California - Berkeley-** http://boinc.berkeley.edu/

7 Conclusions

We have made a proposal to run parallel Data Mining Algorithms, such as parallel decision Trees, parallel Association Rules and parallel Inductive Logic Programming systems in a fault-tolerant setting. To achieve the desired fault-tolerant features we have extended the HARVARD system and have adapted the IndLog ILP system. The HARVARD system was extended with the possibility of including at run-time new tasks to schedule. In the ILP system a new module was encoded to communicate with the HARVARD Master node to suggest new tasks in the format of the task description language the the HARVARD system recognises.

The extended HARVARD system and the Inductive Logic Programming system IndLog were used to analyse an eighty million credit scoring dataset. The fault-tolerant features proved useful when simulating several failures on the machines during the analysis process.

References

1. J. Han and M. Kamber.:Data Mining: Concepts and Techniques. Morgan-Kaufmann Publishers (2001)
2. Kargupta,H. and Chan, P.:Advances in Distributed and Parallel Knowledge Discovery. AAAI/MIT Press (2000)
3. Nuno Amado and Joao Gama and Fernando M. A. Silva.:Parallel Implementation of Decision Tree Learning Algorithms. In: EPIA '01: Proceedings of the 10th Portuguese Conference on Artificial Intelligence on Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving. Springer-Verlag (2001) 6–13
4. R. Agrawal and J. C. Shafer.: Parallel mining of association rules. In: IEEE Trans. On Knowledge And Data Engineering, Vol. 8. (1996) 962–969
5. Nuno A. Fonseca and Fernando Silva and Rui Camacho.: Strategies to Parallelize ILP Systems. In: Proceedings of the 15th International Conference on Inductive Logic Programming (ILP 2005),LNAI, Vol. 3625. Springer-Verlag (2005)
6. Nuno A. Fonseca.: Parallelism in Inductive Logic Programming Systems. University of Porto. Porto (2006)
7. Ruy Ramos, Rui Camacho and Pedro Souto.: A commodity platform for Distributed Data Mining – the HARVARD System. In: 6th Industrial Conference on Data Mining (ICDM 2006), Leipzig - Germany (2006)
8. M. J. Litzkow and M. Livny and M. W. Mutka.: Condor—A Hunter of Idle Workstations. In: Proceedings of the 8th International Conference on Distributed Computing Systems. San Jose - California (1988) 104–111
9. Rui Camacho.: IndLog - Induction in Logic. In: JELIA 2004 - 9th European Conference on Logics in Artificial Intelligence, José Alferes e João Leite (Eds), Springer-Verlag LNAI Vol. 3229. Portugal (2004) 718–721
10. Muggleton, Stephen.: Inductive Logic Programming. In: New Generation Computing, Vol. 8. (1991) 295–318
11. Muggleton, S. and De Raedt, L.: Inductive Logic Programming: Theory and Methods. In: Journal of Logic Programming, Vol. 19/20. (1994) 629–679

12. Ingo Mierswa and Michael Wurst and Ralf Klinkenberg and Martin Scholz and Timm Euler.: YALE: rapid prototyping for complex data mining tasks. In: KDD '06: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2006) 935–940
13. Rui Camacho.: Inducing Models of Human Control Skills using Machine Learning Algorithms. PhD thesis, Faculty of Engineering, University of Porto, Porto - Portugal (2000)
14. I. Foster and C. Kesselman.:The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufmann Publishers (1999) 259–278
15. I. Foster and C. Kesselman.: Globus: A Metacomputing Infrastructure Toolkit. In: International Journal of Supercomputer Applications, Vol. 11(2), (1997) 115–128
16. Ian T. Foster and Carl Kesselman and Steven Tuecke.: The Anatomy of the Grid - Enabling Scalable Virtual Organizations. CoRR, Vol. cs.AR/0103025, (2001)
17. David P. Anderson.: BOINC: A System for Public-Resource Computing and Storage. In: Proceedings on Fifth IEEE/ACM International Workshop on Grid Computing (2004) 4–10
18. I. Foster and C. Kesselman.: The Grid: Blueprint for a New Computing Infrastructure. Morgan-Kaufmann Publishers (1999) 259–278
19. Mario Cannataro and Domenico Talia.: The Knowledge Grid. In: Communications of the ACM, Vol. 46, n.1. (2003) 89–93
20. M. J. Litzkow and M. Livny and M. W. Mutka.: Condor—A Hunter of Idle Workstations. In: Proceedings of the 8th International Conference on Distributed Computing Systems (1988) 104–111