Abstract. Systems performing Data Mining analysis are usually dedicated and expensive. They often require special purpose machines to run the data analysis tool. In this paper we propose an architecture for distributed Data Mining running on general purpose desktop computers. The proposed architecture was deployed in the HARvesting Architectue of idle machines foR Data mining (HARVARD) system. The Harvard system has the following features. Does not require special purpose or expensive machines as it runs in general purpose PCs. It is based on distributed computing using a set of PCs connected in a network. In a Condor fashion it takes advantage of a distributed setting of available and idle computational resources and is adequate for problems that may be decomposed into coarse grain subtasks. The system includes a dynamic updating of the computational resources. It is written in Java and therefore runs on several different platforms that include Linux and Windows. It has fault-tolerant features that make it quite reliable. It may use a wide variety of data analysis tools without modification since it is independent of the data analysis tool. It uses a easy but powerful task specification and control language.

The HARVARD system was deployed using two data analysis tools. A Decision tree tool called C4.5 and an Inductive Logic Programming (ILP) tool.

keywords: Data Processing, Parallel, Distributed Computing, Problem Solving Environments

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 designed efficient data warehouses to store such amounts of data. This is happening not only in Science but mainly in industry. 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 useful information from large amounts of data (data warehouses). However, KDD is facing nowadays two major problems. The amounts of data are becoming 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 located databases. These two problems prompted for a new area of research called Distributed and Parallel Data Mining. This new area addresses the problem of analysing distributed databases and/or making the analysis in a distributed computing setting.

This paper reports on the development and deployment of a computational distributed system capable of extracting knowledge from (very) large amounts of data using techniques of Data Mining (DM) based on Machine Learning (ML) algorithms. The system developed was designed to accommodate easily three of the main stages of a KDD process: pre-processing, Data Mining and post-processing. The system enables the use of different pre and post-processing tasks and the use of different Data Mining tools without any change to it.

Our proposal envisages the following objectives. To provide the user with a simple but powerful language to specify the tasks in the main stages of the KDD process of data analysis. To allow the use of common personal computers in the data analysis process. The computational system uses only idle resources in the organisation. The system will run on a large variety of platforms (Windows and Linux at least) and use parallel and distributed computation. It may run in a cluster or grid environment. The system may use data distributed among several physically separated databases. The system is independent of the data analysis (ML) tool. It has facilities to monitor the KDD process and facilities to recover from major system faults.

Fulfilling the above objectives will have the following advantages. The user may easily configure a data analysis process adequate for his specific needs. The analysis will be affordable to a wide range of organisations since the costs involved are quite low — the machines used are common desktop machines. The analysis process does not disturb the normal work of the organisation since it only uses idle computational resources. A large number of organisations may use it since it runs of a variety of platforms and accesses data that may be physically distributed among several databases.

To attain the objectives of the project we propose the HARVARD computational system. The system allows the user to describe each task of the KDD process in a XML format and to specify the workflow of their execution in a easy to use specification language. The system runs with a single Master node and an unlimited collection of Slave nodes. Both the Master and the Slaves are programmed in Java. The Slaves may access the data directly in a database (using JDBC) and all necessary software for the data analysis tool via HTTP. As will
be described later with more detail the Master reads the KDD process description and generates a workflow graph that is used by a scheduler. The scheduler uses also information concerning the computational resources available. A Slave node may download the data and data analysis tool, monitor its workload and executes the tasks determined by the master. Information concerning the status of the resources are updated regularly.

The rest of the paper is organised as follows. In the Section ?? we present the proposed architecture. In Section ?? we describe the event-driven working of the architecture. In Section ?? we describe how the sub-tasks of the KDD process may be specified by means of a simple but powerful language. We present the fault-tolerant features of HARVARD in Section ?? . The deployment of the HARVARD system is described in Section ?? . Section ?? compares other projects features with the HARVARD capabilities. We conclude in Section ?? .

2 The Architecture

We consider the KDD process as 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 that is presented in Section ?? . The tasks specification is made using XML. The information concerning an individual task includes the name and location of the tool used in the task, the location of the data to be processed and the computational resources required (platform type, memory and disc needs, etc). An example of such a specification is presented in Figure ?? .

The distributed architecture of the HARVARD system is composed of a Master node and a set of computing nodes called 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 each node type.

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 is assigned to 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 below. When a WU finishes, the results associated with it are stored
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, update and provide information concerning the tasks of the KDD process. The TM module constructs a graph structure representing the workflow of the 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 it 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 each task status.

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 available 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 in order for the system to have an 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 computing resource has a time-table of availability of the resource and the policy of use. This information indicates when the machine is available and under what conditions. The usage conditions may indicate that HARVARD 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 the best match at the moment. This module alerts the TM whenever a task must be rescheduled in two situations: if the machine the task was running in is severely delayed to notify the TM module of its workload and; if the pre-established period of use of the machine is expired\(^1\).

\(^1\) In this case the task running on the machine is terminated.
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 slave to download the task’s 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 TM 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.

A Slave node

A Slave node does the actual data analysis work by running the Data Mining 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 it. 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.

---

2 The ones considered more critical for some reason like training longer execution times.
Each Slave has four modules: the Workload Monitoring (WM); the Worker (WO); the Application Wrapper (WR) 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 fetch 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 Application Wrapper module The WR 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 indicated 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 indicated 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 keeps the system 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 if a user has logged in the machine. In the latter case the Master is informed that the task running will be terminated. The Slave 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.

3 An event-driven implementation

The decomposition into modules according to functionality enabled an easier development of the system. Each module is “self-contained” and implement functionalities like; scheduling, resource management, task management, communication, application program control etc. A major goal in the proposed architecture design is that although there are a lot of modules and threads that execute a
wide range of tasks they should not compete for the CPU unnecessarily. If all the modules and threads were running at the same time the system would be slow and the application program would take much more time to run and return the results. The proposed architecture is designed based on an event-driven and message passing implementation. Each module, on both the Master and Slave nodes, has an internal mailbox. Each module reads its mailbox and processes the messages received. The read operation is blocking and therefore if there are no messages they don’t need to work and are in a waiting state that does not compete for CPU. Whenever a message arrives (an event) the message is processed and its content may require some computations. It is only in this situation that the module uses the CPU. The processing of a message may require some computations and usually involves the exchange of messages with other modules activating this way other functionalities. After processing a message a module executes again a read operation that puts it in the waiting state, not computing for CPU, if there are no messages.

The overall result is that a module runs (uses CPU) only when required otherwise does not compete for CPU.

4 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.

An example of the use of the workflow description language is shown in Figure ???. The example represents a simplified KDD process where the pre-processing is a simple feature subset selection (T1 through T8) using a simplified version of parameter tuning (T3, T4, T6 and T7). After the pre-processing the predictive power of the model is estimated using a 5-fold Cross Validation (T9 through T14). Finally the model is constructed in sub-task T18. All of the TIs are described in a separate file in XML and available to the Task Manager module of the Master node of the HARVARD system. The example is explicitly made simple for illustrative purposes. It illustrates how sequential and parallel executions may be interleaved and how easy is to specify a KDD process.

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 a KDD process. For these frequent tasks we intend to provide an interface (or
Fig. 3. An illustrative simple example of a sub-task workflow description.

macro in the workflow description language) where the user just indicates the task and their parameters. For example: 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 the “unfold” that macro into the corresponding graph structure.

5 Fault-tolerance features

The HARVARD system can recover from the failure of the Master node. During its activity the Master node stores status information in an external Database\(^3\). The stored information is enough to enable any node to restart as a Master node and continue the work from the point where the former Master failed. When starting the system the user may specify that one of the Slave nodes (called backup Master node) may take control should a Master failure occur. In such a case the backup Master receives periodically a “alive message” from the Master. If M\(^4\) consecutive of such alive messages fail the backup Master takes control. It recovers the task status from the Data Base and sends all slaves a

\(^3\) Located on a different machine.

\(^4\) A system parameter
message announcing its new status as Master node.

In a normal execution context each Slave node, registered at the Master, does a periodic “alive confirmation” sending a message to the Resource Manager module of the Master. A Slave node that misses more than $N$ confirmation messages is considered unusable. The task running in a Slave that changes state from running to unusable is marked as requiring rescheduling. It will be assigned another Slave node.

6 Deployment of the HARVARD system

Just to test the feasibility of our approach and not to compare the system’s performance on a specific problem we used a freely available large dataset. We have applied both C4.5 and IndLog on the analysis of an Intrusion Detection dataset. This dataset was part of the KDD 1999 conference challenge and is freely available. It is a good test case because there are 24 types of attacks (24 class values) and 489831 labelled firewall log entries (dataset cases) and 2984154 test cases and 39 attributes that results in 743 MB of data. The data and the status logs where stored in two MySQL databases in separate 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.

To analyse the data we used and ILP algorithm [? ,?] implemented in Prolog called IndLog [? ,?] and a Decision Tree tool called C4.5. To use each of these tools we had to provide scripts to execute the tool, scripts to extract the results from the outputs of the tools and scripts to compare the results and choose the best settings based on those results. In order to be able to use different tools the user has to provide such scripts. We have available such scripts for these two tools (C4.5 and IndLog). We intend to produce such scripts for other popular data analysis tools such Apriori (Association Rules analysis ), CART (decision and regression analysis) etc.

At this stage we successfully tested the execution of the system in Linux and Windows platforms. The fault-tolerant features of the system were tested by simulating the breakdown of the Master node and to establish that a Slave node would interrupt whenever a user logged in to that machine.

7 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 subtasks.

---

5 A system parameter

We now present related projects that include some of the HARVARD objectives but none of them have all the functionalities we implemented in HARVARD. Related work includes the Condor system [1], the Boinc system [2] and the Knowledge Grid [3]. In the rest of this section we present, briefly, the related projects and compare them with the HARVARD system.

Condor operates in a university campus or organisation environment. The system aims at maximising the utilisation of workstations with as little interference as possible between the jobs it schedules and the activities of the people who own the workstations. "Condor is a specialised job and resource management system for computing 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" [4]. 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 [5].

The Boinc (Berkeley Open Infrastructure for Network Computing) 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 DLS line can participate in some project and share its own computer power to solve scientific problem whenever the device is idle. The process is very simple since people interested to participate just instal a client software that connects to a project master server. So, when workstation is idle some tasks may be executing. Some projects like SETI@home, Folding@home use the Boinc platform [6].

The Knowledge Grid is a specialised architecture in data mining tools that uses basic global services from Globus architecture [7]. The architecture design of the Knowledge Grid follows the 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 [8].

Like the Condor and Boinc projects, our architecture uses idle workstations and runs under heterogeneous environments with different operating systems (Linux, Windows, OS-X). Differently from Condor we have a much more powerful description language to specify the KDD process and the system just runs one KDD process a time. The HARVARD system differs from Boinc in the way the client receives information and where to find data and data analysis tool. In

---

7 University of California - Berkeley- http://boinc.berkeley.edu/
HARVARD both data and the data analysis tool can be fetched in a DB using JDBC or in the Web via HTTP. HARVARD has also a much more sophisticated fault-tolerant functionalities.

Different from all the referred systems, our proposal implements a two-level language. A specific semantics for the administration of the data mining process, and other for specification of tasks of distributed processing. While one language is designed for the user to specify the process of the knowledge discovery, the other language is used by the system to manage the distributed computations.

In the like of Globus, the user sees a computation environment for knowledge extraction using algorithms Machine Learning in a virtual computer. In this way an analyst of businessman can use an interface that allows applying techniques of extraction of knowledge in a fast and efficient way without need of taking knowledge of the support platform. It is as if he had interacting in an environment of high performance commutating. The final result is a friendly and interactive platform with the user and efficient in terms of computational resources that makes use of distributed and idle resources.

8 Conclusions

We have proposed an architecture for Distributed Knowledge Discovery in Databases that is capable of using different Data Analysis tools without any modification. The architecture enables the use of general purpose desktop computers that are idle in an organisation. This latter feature makes the Data Mining process affordable to a wider range of organisations and the process of analysing the data does not interfere with the normal work of the organisation. The architecture features also allow the processing of data in distributed data bases and the migration of the data analysis tools to avoid the transfer of large volumes of data.

We have deployed the architecture in the HARVARD system and tested it using a Decision Tree data and Inductive Logic Programming system data analysis tools. We provide a workflow control language that enables the user to easily describe the KDD process with the sub-tasks and options of his choice. The HARVARD system is programmed in Java and may run in a wide range of platforms. It has fault-tolerant features that make it quite reliable.

As future work we envisage to develop to enrich the control description language to easy even more the specification of the Data Mining process by the user.

References