Towards Flexibility in a Distributed Data Mining Framework

José M. Peña
DATSI, Universidad Politécnica de Madrid
Campus de Montegancedo S/N, 28660
Madrid, Spain
jmpena@fi.upm.es

Ernestina Menasalvas
DLSIIS, Universidad Politécnica de Madrid
Campus de Montegancedo S/N, 28660
Madrid, Spain
emenasalvas@fi.upm.es

ABSTRACT

Commercial data mining systems are becoming more and more complex. Advances features like, extensible algorithm toolboxes and programming APIs, allow these systems to include new operations to mine and to prepare data. Although these characteristics are necessary trends performance cannot be compromised. In order to keep the balance between functionality and performance new design strategies are needed. In this paper we present a flexible control architecture for distributed data mining.

1. INTRODUCTION

Distributed data mining (DDM) systems must keep the existing features of the current centralized versions. Nevertheless new features should be added to deal with the new trends:

- New algorithms should be included easily.
- Integration between relational a data mining operations is a main goal.
- State-of-the-art trends in interoperability, like CORBA, should be talen into account.
- Factors like the inclusion of algorithms, changes on system topology, different user privileges or system load restrictions have a deep impact in the overall performance of the system. Important decisions like task scheduling, resource management or partition schema should be tuned again almost every time one of these factors is modified.

Nowadays, there is no universal strategy to configure a distributed data mining environment to be optimal in the resolution of every query. As an alternative our contribution presents an appropriate architecture to support multiple optimization and configuration strategies in data mining systems. This architecture provides a framework in which control policies can be changed to configure and optimize the performance of the system, even while it is running. This will make it possible to tune the system according to the execution requirements such as the application environment, user preferences or resource restrictions.

DMKD Workshop '01 Santa Barbara, California USA

2. CURRENT TRENDS

The evolution of distributed data mining systems is getting pushed by the need to process trillions of records in a restricted time frame. Thus, many new systems have been designed to achieve this objective.

Distributed Data Mining Systems

JAM [11] has been developed to gather information from sparse data sources and induce a global classification model. JAM technology is based on the meta-learning technique [2].
The PADMA system [9] is a document analysis tool working on a distributed environment, based on cooperative agents. It works without any relational database underneath. Instead, there are PADMA agents that perform several relational operations with the information extracted from the documents.
The Papyrus system [6] is able to mine distributed data sources on a WAN network scenario. Papyrus system uses meta-clusters to generate local models which are exchanged to generate a global model. The idea is founded on a theory similar to JAM system, nevertheless they use a model representation language (PMML [5]) and storage system called Osiris.
Kensington [10] architecture is based on a distributed component environment located at different nodes on a generic network, like the Internet. Kensignton provides three kind of components: (i) User oriented components, (ii) Application servers and (iii) Third level servers.

DDM System Taxonomy and Requirements

Distributed and parallel data mining systems are more often experimental systems rather than commercial products. Due to this most of them are focused only on a particular data mining problem. Papyrus and JAM are distributed classifiers. Other problems like association calculation [1, 3, 7] or sequential patterns [12, 13] have also distributed algorithms. Each of this algorithms deals with different factors of: (i) data distribution (horizontal or vertical), (ii) communication schemas (broadcast or unicast), (iii) synchronization and coordination (collaborative, centralized control, voting, ...) or (iv) distributed task granularity.
In order to run a number of these algorithms in a common system many different behaviors should be supported. A

system having this features is by itself a distributed control challenge in which two main factors should be balanced. First, the different algorithms that exist and may be developed in a future and second, the performance of data processing must be efficient.

3. MOIRAE ARCHITECTURE

MOIRAE architecture has been designed to achieve distributed control in complex environments. This is a generic architecture based on the mechanism/policy paradigm.

3.1 Mechanism/Policy Paradigm

Complex applications, like operating systems (OS), that are in charge of resource management have two main goals: efficiency and operational support. Data mining applications are quite similar in many aspects. Efficiency in the execution of programs on a expensive new hardware is as important as the efficient execution of rule induction algorithms in a time-constrained problem. The operational support in an OS ensures the possibility to run very different applications, providing restricted access to I/O devices and managing resources like memory or CPU time. As it has been already mentioned, one of the requirement of modern DDM systems is the extensibility with new algorithms and techniques as easy as possible. In both cases applications and algorithms need specific features that should be provided by the supporting entity (OS or the data mining system) using an open and generic interface.
OS design uses a particular approach to reach these objectives. It is called mechanism/policy (M/P) paradigm. This paradigm claims an analysis of the systems in two levels:

- Operational level: Features provided by the system. This level contains all the actions (operations) performed by the system as well as the functions used to monitor its status and the environment.
- Control level: Decisions that rule the system. This second level defines the control issues applicable to the operational level functions. These rules describe when and how the operations are performed.

3.2 MOIRAE Models

Traditional M/P design describes both elements, operational actions and control decisions, separately but final implementations merge them into a unique system. These systems provide static features for the operational part (mechanisms) and also static control decisions for those mechanisms. Dynamic control configuration requires flexible operations for creating, deleting and updating control rules. In order to support these dynamic control changes a generic architecture is defined. This architecture is called MOIRAE (Management of Operational Interconnected Robots using Agent Engines).
MOIRAE is an abstract architecture that could be implemented with many different technologies, using different programming languages and at many different levels of detail. MOIRAE just provides a framework, a set of ideas and methods, to design a distributed system when flexible control decisions can be controlled externally. MOIRAE architecture is divided into different *Models* that have detailed descriptions of functionalities, restrictions and elements: (i) **Component Model**, (ii) **Relationship Model**, (iii) **Architecture Model** and (iv) **Control Model**.

3.2.1 Component Model

This Model describes the structure that every element of the system must have. MOIRAE architecture provides control features based on the functionalities provided by each of the elements. These atomic elements are called *components*. Following the M/C paradigm each component defines two kinds of functions, operational functions and control functions. Each of these groups of functions is implemented in a *plane*. The *Operational Plane* includes all the modules that provide the operational functions. The *Control Plane* provides the features necessary to control the other *plane* as well as to interact with other *Control Planes*.

- **Operational Plane**: These elements provide all the functions required to achieve the work performed by the component. The functions, their design or implementation and the exact elements included in the plane depend on the task performed by the component.
  The responsibilities of this plane heavily depend on the functions of the component.
- **Control Plane**: The elements of this plane interact either with the control plane from other components or with the operational part of their own component. The control plane solves complex situations called *conflicts*, sending orders to the operational plane.
  The control plane plays the role of an intelligent agent. It perceives the environment (situation of the operational plane), communicates with other agents (control planes from other components) and interacts with its immediate world. The control plane may also be referred as *control agent*.

3.2.2 Relationship Model

This Model defines a taxonomy of the possible communication schemas between two or among more components. This classification assigns values to each relationship for cardinality of the members, scope of the relationship and addressing methods. The relationship characteristics are: Cardinality (One–to–one, one–to–many and many–to–many), Scope (public vs private relationships), Addressing (identified vs anonymous).

3.2.3 Architecture Model

Based on the concepts provided by the Component Model and Relationship Model, the Architecture Model describes how multiple components are combined and interconnected to deal with the tasks performed by any specific system. The network of interconnected components is called *interaction graph*. This *graph* defines the components and their relationships. For every system, there are two different *graphs*: *operational graph* and *control graph*. The *operational graph* shows the relations used by the operational planes of the components. This *graph* depends on the task performed by the components and the services provided by the system. MOIRAE does not define any specification on this *graph*. *Control graphs* represent the control relationships among the components. This Model provides a generic schema of this *graph*. Control interactions are based on a hierarchical organization.

3.2.4 Control Model

The last Model of the architecture is the Control Model. This Model shows how control decisions are taken either locally or as a contribution of different control planes. When the control plane is activated (when a conflict is detected), the agent evaluates the alternatives to solve the problem. As a result, the control plane returns a sequence of actions to be performed to solve the conflict. For complex problems the control plane would be unable to achieve an appropriate solution by itself. Control Model specifies three different control actions that rule the cooperative solution of complex problems:

- **Control Propagation**: When a control plane is unable to solve a problem it submits the problem description (e.g.: the conflict) and any additional information to the control plane immediately superior in the hierarchy.
- **Control Delegation**: After receiving a control propagation from a lower element, the upper element may take three different alternatives:
  1. If it is also unable to solve the problem it propagates up the conflict as well.
  2. If it can solve the problem, it may reply to the original component with the sequence of actions necessary to solve the problem. This original component executes these actions.
  3. In the last situation it is also possible that the component, instead of replying with the sequence of actions the component may provide the information necessary to solve the problem in the lower component. This information could be used also in any future situation. This alternative is called *Control Delegation*.
- **Control Revoke**: This action is the opposite to the *control delegation* one. Using this control action any upper component in the hierarchy may delete information from the agent of any lower element. This action may be executed anytime and not only as a response of a *control propagation*.

4. MOIRAE IN DATA MINING

MOIRAE architectures is an abstract tool to develop distributed systems of any kind. A distributed data mining system called DIDAMISYS has been designed using this architecture. The system has been divided into different federations. The federations are groups of components that collaborate to achieve a well-defined set of functionalities. DIDAMISYS has the following federations:

- Interfaces Federation: Components that interact with the user.
- Data Mining Federation: Defines data mining queries, describing both the algorithms to be used and the data sources to be accessed.
- Data Warehouse Federation: Defines data administration tasks, like data integration, retrieval or information gathering.
- Data & Process Federation: Achieves the sequence of operations defined either by the Data Mining Federation or by the Data Warehouse Federation.

A data mining algorithm or any administrative task can be split into a sequences of basic operations. Data Mining and Data Warehouse Federations only prepare execution plans that are sequences of these basic operations. Nevertheless, it is the Data & Process Federation the one that deals with the main data processing and mining tasks. As a consequence the main work load of the system is associated to this federation. Due to this the control policies are mainly focused on the components of this federation. In what follows the Data & Process Federation is explained in detail.

Sketch of a Solution

In the distributed data mining system architecture that we propose the main work load is due to the tasks developed by the components of the Data & Process Federation. Thus, this federation obtains the maximum integration between processes executing in the system and the data they use is obtained. It is the most internal federation of the system in the sense that interfaces do not access directly to it for requesting its services. Only data mining and data warehouse federations are able to submit operations to these components.

In order to provide its features this federation design is based on a pair of logical buses. These buses interconnect several system nodes and allow different components to migrate from one node to another: (i) **Process Component Bus**: This bus supports operator migration between system nodes. Each operator is programmed as a different component and they can be run on any of the machines belonging to this bus and (ii) **Data Component Bus**: This other bus allows data replication and migration between the nodes. Each data table stored on a node is represented as a component on this bus. If the component is moved or copied to another node on the bus, the data it represents are transmitted over the network via this bus. Data migration is not one of the actions the system should do during algorithm execution and it is planned to be performed as tuning tasks. In order to succeed in the execution of the processes this federation performs the following internal functions:

- *Component distribution*: When a request for a service enters in this federation it always comes along with a *group description*. This group description define the group of operators and the tables that must be included in the chain to be executed. Taking into account this information, it has to be decided the most appropriate distribution of the chain in the available nodes in order to ensure an optimal execution of the chain. To make the decision of the location of source tables, the amount of available memory and the available operators have to be taken into account.
- *Operator load*: An instance of any operator belonging to a chain of operators to be installed has to be started in the chosen machine. The information required in order to maintain the bus of processes is stored in what has been called *operator cache*.
- *Execution control*: While the chain is being installed,

it is also required to build certain control components that are responsible for the proper execution of the chain, this is to say that they must ensure the correct synchronization when executing operators.

- *Data representation and physical storage*: The data bus has to be managed to offer a quick access to data tables. The design of the internal structure of the tables as well as the way in which they are stored in secondary devices are quite related to each other in order to obtain the optimal access. Data stored on the physical media is loaded into memory by the components on this bus and then these components act as a server of the original data table they represent. The amount of memory used to load the data is constrained by the management component of this bus balancing resource usage by different components. If the memory space assigned to one of these components is not enough to load all the data then a swapping process is executed to load and release data blocks between memory and disk. These data blocks are called *pages*.

When a process is submitted to this federation from either Data Mining (algorithm executions) or Data Warehousing (data management/integration tasks) Federations, it is represented as *group description*. This component defines which operator components must be executed and which data sources participate. Both operator and data components are activated, hosting them on specific nodes, once active operators and data components are linked (by their operational plane).
While their execution, external factors or control strategies might require to move the components from one node to another. Moving an operator may cause previous local links to become remote. Remote and local links provide the same features but it is obvious that remote data access require data transfer over the network, due to this strategic location of operations as near as possible to the data they use. Data components can also be moved but they always require original data to be moved also, this means for large data tables a lot of time therefore it is better to plan these actions when the system is idle and only for the most common used tables.

5. CONCLUSIONS

This paper has introduced MOIRAE, a generic architecture based on M/P paradigm. MOIRAE architecture has been used as a foundation of a new distributed data mining system called DIDAMISYS. DIDAMISYS system is the distributed version of an existing data mining system called DAMISYS [4]. It has been also presented the main DIDDAMISYS federations, highlighting the role of Data & Process Federation. The design of this federation provides the innovative point of this system. Load balancing issues, flexible optimization, resource management and fault tolerance are supported by this federation.

6. REFERENCES

[1] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *Ieee Trans. On Knowledge And Data Engineering*, 8:962–969, 1996.

[2] Philip K. Chan and Salvatore J. Stolfo. Sharing learned models among remote database partitions by local meta-learning. page 2. AAAI Press, 1996.

[3] D. W. Cheung, V. T. Ng, A. W. Fu, and Y. J. Fu. Efficient mining of association rules in distributed databases. *Ieee Trans. On Knowledge And Data Engineering*, 8:911–922, December 1996.

[4] Covadonga Fernández, José M. Peña, Juan F. Martínez, Óscar Delgado, J. Ignacio López, M. Ángeles Luna, and J.F. Borja Pardo. DAMISYS: An overview. In *Proceedings of DAWAK'99, Florencia, Italy*, pages 224–230, August 1999.

[5] R. L. Grossman, S. Bailey, A. Ramu, B. Malhi, P. Hallstrom, I. Pulleyn, and X. Qin. The management and mining of multiple predictive models using the predictive modeling markup language (PMML). *Information and Software Technology*, 1999.

[6] R. L. Grossman, S. Kasif, D. Mon, A. Ramu, and B. Malhi. The preliminary design of Papyrus: A system for high performance, distributed data mining over clusters, meta-clusters and super-clusters. In *Proceedings of the KDD-98 Workshop on Distributed Data Mining*. AAAI Press, 1998.

[7] E.H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *Proceedings ACM SIGMOD Conference for Management of Data*, May 1997.

[8] David Heckerman, Heikki Mannila, Daryl Pregibon, and Ramasamy Uthurusamy, editors. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD-97)*. AAAI Press, 1997.

[9] Hillol Kargupta, Ilker Hamzaoglu, and Brian Stafford. Scalable, distributed data mining-an agent architecture. In Heckerman et al. [8], page 211.

[10] Kensingston, Enterprise Data Mining. Kensington: New generation enterprise data mining. White Paper, 1999. Parallel Computing Research Centre, Department of Computing Imperial College, (Contact Martin Khler).

[11] A. Prodromidis, P. Chan, and S. Stolfo. chapter Meta-learning in distributed data mining systems: Issues and approaches. AAAI/MIT Press, 2000.

[12] T. Shintani and M. Kitsuregawa. Mining algorithms for sequential patterns in parallel: Hash based approach. In *2nd Pacific-Asian Conference on Knowledge Discovery and Data Mining*, Appril 1998.

[13] M. J. Zaki. *Large-Scale Parallel Data Mining*, chapter Parallel Sequence Mining on SMP Machines. 2000. Published in [14].

[14] M. J. Zaki and C. T. Ho, editors. *Large-Scale Parallel Data Mining*, volume 1759 of *LNCS*. Springer-Verlag, 2000.