Using a Problem Solving Method for Dynamic Configuration of Data Mining Applications on the Grid

Francisco Flávio de Souza, Vasco Furtado

Universidade de Fortaleza - UNIFOR,
Mestrado em Informática Aplicada – MIA
Washington Soares, 1521, Fortaleza – Ce, Brazil
fsouza@uol.com.br, vasco@unifor.br

Abstract. In this paper, we describe how SMARTBASEG, a knowledge-based architecture, can optimize dynamically data mining (DM) applications for the Grid. SMARTBASEG can be seen as a layer that intercepts the submission of a DM application to the Grid, identifies the procedures to be paralleled and optimizes the composition of the job to be submitted to the Grid. The job composition process is done by means of a problem solving method (PSM) of configuration that adopts a propose-and-revise strategy. We present our approach by describing formally the implementation of the PSM in SMARTBASEG and by exemplifying the representation of heuristics for DM application optimization by means of constraints and repairs. Finally, we show the preliminary results we have obtained in some experiments.

1 Introduction

The process of Knowledge Discovery in Database (KDD) for extracting useful information from data requires high processing capability and is often seen as a potential application domain to the Grid [1]. The performance of KDD phases is improved with task parallelization through the several grid nodes. Typically, KDD tasks compose a job that is submitted to a scheduler, which is responsible for finding the adequate nodes to perform the tasks.

The task preparation for composing a job as well as the scheduling problem has been studied in the Grid, and various approaches have appeared. Recently some researchers have tried to use artificial intelligence techniques in this context by applying planning algorithms to create a task plan taking into account the features of the application (an algorithm’s implementation) and the Grid as well [2]. These approaches have the basic assumption that is possible to prepare an efficient plan in advance, which is still static during the whole KDD process. The plan is prepared with basis on information about the application that comes typically from an estimative of the job execution time based on historical data and on information about the Grid. Despite the generality of these approaches, they are still suffering of problems of load balancing, mainly because precise information derived from historical data about the application can be hard to converge and then the information is inaccurate during the job process-
Moreover, the dynamic nature of the grid nodes changes on the plan initially prepared, which is not trivial to do.

In the context of KDD and mainly in data mining (DM) phase, the task composition problem assumes particular contours. First of all, it is important to observe that the inner DM procedures (routines or parts of applications) that compose a DM application and are potential candidates to compose a job of tasks to be submitted to the Grid are very prone to load unbalancing [3]. Applications of Top-Down Induction Decision Tree (TDIDT) algorithms – one of the most popular classification algorithms – represent an example of that. They are composed of iterative phases to explore databases, which are gradually being divided but hardly in equal sizes. In a general way, the DM tasks must be treated in a very fine-grained way being inefficient to prepare a set of tasks in advance to be submitted to the Grid.

The good news is that DM algorithms follow a structure that is being more and more formalized. Such formalization improves the development of DM applications through reuse and increments the level of portability of the applications. The JDM (Java Data Mining) library is an example [4]. Based on this, it is possible to define some basic features of the application in order to use them during the job composition phase for optimizing the application execution.

In this work, we build on early works on DM formalization such as the Knowledge Grid [5] and the SMARTBASEG architecture [6], which represents knowledge about DM in ontology. SMARTBASEG proposes the creation of a layer for separating concerns related to software development to the Grid, such as the optimization aspect leaving the developer of DM applications free of concerns related to the development of grid-dependent procedures. In this paper, we describe how this layer can optimize dynamically DM applications for the Grid. It is implemented as a set of services that intercept the submission of a DM application to the Grid, identify the procedures to be paralleled and optimize the composition of the job to be sent to the Grid. The job composition process is done by means of a problem solving method (PSM) of configuration that adopts a propose-and-revise strategy [7]. Following this approach, the developer of DM applications is free of concerns as for optimizations of his/her application as well as for grid middleware peculiarities. Furthermore, the knowledge about job composition and optimization is represented explicitly in terms of constraints and repairs being possible to be updated by an expert/administrator as long as DM applications are developed following this approach.

We present our approach by describing the implementation of the PSM in SMARTBASEG and by exemplifying the representation of heuristics for DM application optimization by means of constraints and repairs. The implementation was done in OurGrid [8], a grid middleware developed for running Bag-of-Tasks (BoT) applications (the ones with independent tasks). The choice of OurGrid as platform incorporates a subjacent goal. The main characteristic of OurGrid is that it is very easy to be deployed because it doesn’t require any initial configuration of the grid nodes. In this context, pre-defined plans that require initial preparation of the grid nodes are undesirable. Thus, the optimization aspects must be treated in a run-time way.
2 Related Works

Most of the approaches to DM in Grids have problems of efficiency [9]. In particular, a BoT Grid provokes overhead caused by processes performed on central machines and then issues of load balancing can be crucial. On the other hand, some works aim at making the DM phase more efficient through parallel access to the databases being mined. These solutions are typically found in database management systems, which are already available in commercial products [10]. However, these works do not consider the fact that the increase in DM efficiency does not depend exclusively on the improvement of data access. In fact, the complexity of a procedure is connected to its own logic of data handling. For example, a procedure that sorts a file requires more resources than another one that simply reads the same file. Therefore, the lack of knowledge by a scheduler about the processing type and complexity makes an intelligent choice of grid nodes unfeasible. Another sub-group of works on DM suggests the implementation of heuristics in algorithms and/or in schedulers to deal with situations characteristic of a heterogeneous environment, such as the Grid [11]. The strategy is to identify the best grid configuration for a given application. They mainly consider the granularity of tasks, the size of the database and the quantity of nodes to be used, which provides a basis for deciding the best way to schedule and distribute tasks. The identification of these properties is possible through the use of grid monitoring mechanisms and algorithms for the learning process that dynamically determines the state of the grid nodes [12]. With this approach, it is possible to make an intelligent scheduling of mining tasks. These works have greatly influenced our proposal.

In [2], there is a description of a system that generates executable grid workflows given high-level specification of desired results. It uses Artificial Intelligence planning techniques to compose a workflow of the application execution. The authors consider that it is possible to select application components and computing resources before running the application. Doing so, a plan is automatically generated.

3 The SMARTBASEG Architecture

SMARTBASEG is a software architecture for helping the development and optimization (in terms of performance) of DM applications in Grids [6]. It considers that the task scheduling and load balancing problems for DM procedures must be solved not only with the search for information about the Grid’s state. It considers that DM applications can be characterized in terms of ontology (that describes concepts involved in the steps that compose DM algorithms) and that this characterization enables the definition of an optimization layer that can dynamically choose the best way to submit DM procedures to the Grid.

The architecture provides a set of components to be used by a DM application developer. The components of DM applications encapsulate grid concepts and uses components that implement DM functionalities based on JDM – the Sun JSR 073 specification for DM [4]. They also encapsulate which procedures will be sent to the Grid in parallel. The developer of a DM application can access these components and take the advantage of code reuse and a higher-level abstraction of the grid concept.
The overall goal is to obtain higher quality in the development and use of these applications, independent of grid configuration and the application itself. The architecture provides components that achieve this transparency (during application development) and also a service for tasks optimization (at application execution time).

The service for tasks optimization is responsible for optimizing the performance of an application being executed. It applies heuristics based on the knowledge of an expert to prepare jobs/tasks to be submitted to the Grid. It uses information about the characteristics of DM applications and of Grids. In this strategy, there is a knowledge base (containing rules that explicit heuristics) that enables the choice of appropriate dynamic task configuration. For instance, some tasks of a job (that perform some application procedures) can be grouped, thus, generating a new job that has fewer tasks (where each new task is composed of two or more original ones) in order to produce a better load balancing in the grid nodes.

The ontology used by SMARTBASEG has concepts about DM applications (based on [13]) and Grids. For instance, the definition that C4.5 is a supervised TDIDT classification algorithm is done in the DM ontology, as well as the characteristics of the procedures that compose this algorithm. Moreover, grid properties as the fact that a certain grid middleware only executes BoT applications are made explicit in the ontology. These concepts facilitate the generation of rules in the knowledge base to be explored by the optimization layer during the execution of DM applications. Besides the grid and DM concepts, in the ontology are also represented the concepts that were defined with the goal to integrate concepts from these two domains. The concepts of process (an application in execution) and transaction (each interaction made by a process with the Grid in order to get a set of DM procedures executed by it) are examples of this.

Fig. 1. A part of SMARTBASEG’s ontology describing DM and grid concepts
In Figure 1, we have a part of an implemented ontology, its taxonomies (solid lines) and how its concepts relate to each other (traced lines). Particularly, we exemplify the definition of the concepts involved in a DM application, in this case, the C45App01. In the ontology, it is described as software that implements the C4.5 algorithm, which is a TDIDT algorithm and belongs to the classification function. We can see that the C45App01 has the procedures C45AttEvalC01 and C45AttEvalD01, responsible for the continuous and discrete attribute evaluation, according to C4.5 algorithm. Basically, these procedures realize the same activity but as they have different computing complexity, they are represented particularly. On the same figure, it’s possible to note that the concepts of process and transaction perform the integration between the concepts of application and procedures from the DM domain with the concepts of job and task from the grid domain. Finally, we can see that the concept of grid is declared as an entity that has nodes to perform tasks and a middleware for its own management and the schedule of tasks. As an example, we have a grid middleware that deals with BoT applications, the Ourgrid. The concepts described in the ontology are used by SMARTBASEG that adopts a propose-and-revise strategy to configure jobs to be submitted to the Grid. In the next section, we describe how this occurs.

4 Configuring Grid-Adapted Tasks

4.1 Formalizing the job and task configuration problem

In a general sense, we can say that one of the main problems SMARTBASEG proposes to solve is the translation of a set of DM procedures into a set of tasks configured and optimized for grid execution. In this context, we assume that the computational nodes of the grid are roughly equivalent and that they can execute any DM procedure. The number of available nodes at a given moment is the crucial piece of information when defining this set of tasks, referred to here as a job. Therefore, we can formally describe the above problem as a relation ($\mathcal{R}$), where $\mathcal{R} = (A, B, \mathcal{P}(x, y))$, such that $A = N_t \times E \times P_s$ and $B = J_s$, where:

- $N_t$: the set of possible quantities of available grid nodes at a given moment $t$, where $N_t \neq \emptyset$ and $\forall n \in N_t \Rightarrow n > 0$;
- $P_t$: the set of all the DM procedures that can be submitted to the Grid by an application at a given moment $t$, where $P_t \neq \emptyset$;
- $P_s$: the set of all the possible submissions of DM procedures to the grid at a given moment $t$, meaning $P_s = \{P \mid P = \{p \mid p \in P_t\}\}$, where $P_s \neq \emptyset$;
- $f_e$: function that estimates the execution time of each DM procedure that can be submitted to the Grid, meaning $f_e: P_t \rightarrow \mathbb{R}^+$, such that if $p \in P_t$, $f_e(p)$ indicates its estimate of resource consumption for a single grid node and if $p_i$ and $p_j$ are in $P_t$ and $f_e(p_i) > f_e(p_j)$, $p_i$ will consume more node resources than $p_j$;
- $E = \{f_e \mid f_e \text{ has the same properties of } f_e \text{ and vice-versa}\}$, where $E \neq \emptyset$;
- $T_s$: the set of all the tasks or minimal units of work assigned to a single grid node, where each one represents one or more DM procedures that can be submitted to the Grid at a given moment $t$, or rather $T_s = \{T \mid T \neq \emptyset \text{ and } T \subset P\}$, such that $P \in P_s$;
- $J_s$: the set of all the jobs (or groups of tasks) that can be submitted to the Grid at a given moment $t$, or rather $J_s = \{ J \mid J \neq \{ \emptyset \} \}$ and $J \subset T_s$ such that $\forall T_p \in J$ and $\forall T_q \in J$, where $p \neq q \Rightarrow T_p \cap T_q = \emptyset$, $T_p \subset P$ and $T_q \subset P$, such that $P \in Ps$;
- $f_w$: a function of a task’s work load, or rather $f_w(T) = \sum_{i=1}^{n} |T| f_e(p_i)$, such that $p_i \in T$, where $T \in T_s$;
- $W_s$: a set of possible jobs whose individual tasks represent approximately the same workload, or rather $W_s = \{ W \in J_s \mid \forall T_i \in W \text{ and } \forall T_j \in W, \text{ where } i \neq j \Rightarrow f_w(T_i) \equiv f_w(T_j) \}$, where $W_s \subset J_s$ and $W_s \neq \{ \emptyset \}$;
- $|W|$: the number of tasks of the configured job, where $W \in W_s$, which is defined by $n$ (the number of nodes at a given moment $t$) and $|P|$ (the number of DM procedures of an application at a given moment $t$), where $P \in Ps$, such that $1 \leq |W| \leq \min(n, |P|)$;
- $\varphi(x, y)$: “$y$ is the result of an optimized job configuration based on $x$”, or rather if $x = (n, f_e, P)$, where $n \in \mathbb{N}$, $f_e \in E$ and $P \in Ps$, and $y = W$, where $W \in W_s$ then $\varphi(x, y) = true$. On the other hand, if $y = Q$, where $Q \notin W_s$ then $\varphi(x, y) = false$. It is important to note that there are cases in which optimized load-balancing is unattainable using the criteria we present. For example, suppose $n = 2$ (where $n \in \mathbb{N}$) and $P = \{ p_1, p_2 \}$ (where $P \in Ps$), such that $f_e(p_1) > f_e(p_2)$, then, for any combination of tasks that involve the 2 nodes, one of them will have a bigger work load than the other one’s (see formulas 1 and 2). A way around this would be to define only one task $T = P$ and assign it to any node of set $N$, so that the load-balancing definition would be satisfied. However, this would come with a performance cost for the parent application of $p_1$ or $p_2$, or the throughput of the grid would be negatively impacted if they belonged to different applications.

4.2 The job and task configuration problem in SMARTBaseG.

In SMARTBaseG, we did a few simplifications/adaptations of the configuration process in order to implement an architectural prototype and collect our initial results:
- The set $P_t$, and consequently $P$ and $Ps$, contain only DM procedures that can be submitted to the Grid by an application in a given moment $t$. Even though multiple applications can submit procedures to the Grid at the same time, the configuration of their tasks will be done at different moments;
- It was adopted $|W| = \min(n, |P|)$, where $W$ is the job configured by SMARTBaseG;
- Regarding the execution time estimate of a DM procedure ($f_e$), we decided to adopt the complexity of its algorithm as an indicator. This was simple because this is a DM application characteristic stored in SMARTBaseG’s ontology. If a procedure has complexity $O(n)$ and another procedure has $O(n\log n)$, and both have an input dataset of the same size, we can speculate that the first will execute more quickly than the second, although it is not guaranteed. If the dataset input size of the second is smaller than that of the first, then this prediction is less reliable.
However, SMARTBaseG’s ontology and rule set allows other prediction strategies to be developed. Using information about DM applications (and the Grid), it is possible to cover large numbers of permutations and achieve efficient load balancing in a broader range of situations.

Using these definitions we were able to model the job configuration algorithm called propose-and-revise. This algorithm is a PSM for configuration design tasks originated from works on knowledge acquisition. The basic goal was to define models that aid the process of expertise modeling. A configuration problem is described as a set of input and output parameters (or variables), a set of constraints, and a set of boundaries to detect and fix constraint violations. A solution consists of a value assigned to the output parameters that does not violate any constraint [7]. In our case, the inputs (or components) are the DM procedures to be submitted to the Grid (which was not assigned to a task yet) and a set of empty tasks. On the other hand, the output (or the job design) is a set, initially empty, that will receive all the components, one by one. At the final of the task configuration, the design becomes a set of balanced tasks composed of DM procedures. The main activity is to configure the design by successive proposals of a new component and verifying if any constraint that defines the quality of the configuration was violated. In that case, repairs are proposed and the activity restarts.

Table 1. Examples of constraints to be verified after each propose step

<table>
<thead>
<tr>
<th>Constr.</th>
<th>Constraint Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>If (Ci ∈ Ts) and ((∃ Cp ∈ P and Not ∃Ck ∈ Ts, such that Cp ∈ Ck) or (∃Ck ∈ Ts, such that</td>
</tr>
<tr>
<td>2</td>
<td>If (Ci ∈ P) and (Ck ∈ Ts) then ∃Ck ∈ Ts, such that Ck ∈ Ck;</td>
</tr>
<tr>
<td>3</td>
<td>If (Ci ∈ Ts) and (Cj ∈ Ts) and (∀p ∈ Ci ⇒ p is-a “TDIDT Procedure”) and (∀q ∈ Cj ⇒ q is-a “TDIDT Procedure”) then</td>
</tr>
<tr>
<td>4</td>
<td>If (Ci ∈ Ts) and (Cj ∈ Ts) and (∀p ∈ Ci ⇒ p is-a “TDIDT Procedure”) and (∀q ∈ Cj ⇒ q is-a “TDIDT Procedure”) then</td>
</tr>
</tbody>
</table>

The next step was to define the constraints that would regulate the process and guarantee its effectiveness. We begin by setting down some basic constraints in table 1, where Ci is a component being proposed to the design (or that violated a constraint) at a given moment, and Cj, Ck and Cp are some components already inside the design:

- Every task must contain at least one DM procedure (constraint 1);
- Every DM procedure must be assigned to one, and only one task (constraint 2).

Once these constraints were established, we moved into the DM domain, more specifically TDIDT, and created two more constraints. Constraint number 3 tries to balance the number of procedures amongst the tasks. Constraint number 4 makes specific adjustments regarding the computational effort by taking into account the nature of the DM procedure represented by “Cont. Attrib. Evaluation”.

In table 2, we see the possible courses of action taken in case any of the aforementioned constraints are violated. The codes in Violation column correspond to the constraints of table 1 that were violated after the inclusion of some component. The Fixes
column details the appropriate actions taken to repair the violations. For violation 1, there are two repair options: use a DM procedure that has not yet been assigned, or transfer a DM procedure from a task with at least two procedures to an empty task. For violation 2, the unassigned DM procedure is given to the task containing the smallest number of procedures. In the case of violation 3, the task assigned to the most procedures is compared to that with the fewest. The difference between their respective numbers of procedures is halved and that number of procedures is transferred to the task with the fewest procedures. This is done to balance the computational effort amongst them. Finally, in violation 4, we have the following situation: given two tasks where the first and second are, respectively, the ones that represent the largest and smallest quantity of the procedure “Cont. Attrib. Evaluation”, half of the excess of the first task will be transferred to the second.

Table 2. Actions list to fix constraint violations

<table>
<thead>
<tr>
<th>Violation</th>
<th>Fix List</th>
</tr>
</thead>
</table>
| 1 | \[C_i =: \{C_p\}, \text{ where } C_p \in P; \text{ and } \not \exists C_k \in Ts, \text{ such that } C_p \in C_k; \text{ or } \]
| | \[C_j =: C_j - \{C_p\} \text{ and } C_i =: \{C_p\}, \text{ where } C_p \in P; C_j \in Ts \text{ and } |C_j| \geq 2; \text{ and } \not \exists C_k \in Ts, \text{ such that } |C_k| > |C_j| \] |
| 2 | \[C_i =: C_j \cup \{C_i\}, \text{ where } C_j \in Ts \text{ and } \not \exists C_k \in Ts, \text{ such that } |C_i| < |C_j| \] |
| 3 | \[C_i =: C_j - C_d \text{ and } C_i =: C_i \cup C_d, \text{ where } \not \exists C_k \in Ts, \text{ such that } |C_i| > |C_j|, C_d \subset C_j \text{ and } |C_d| = (|C_j| - |C_i|)/2 \] |
| 4 | \[C_j =: C_j - C_d \text{ and } C_i =: C_i \cup C_d, \text{ where } \not \exists C_k \in Ts, \text{ such that } |Z| > |Y|, \text{ where } Z = \{z \mid z \in C_d \text{ and } z \text{ is-a “Cont. Attrib. Evaluation”}\} \text{ and } Y = \{y \mid y \in C_j \text{ and } y \text{ is-a “Cont. Attrib. Evaluation”}\}; \text{ and } C_d \subset Y \text{ and } |C_d| = (|Y| - |X|)/2, \text{ where } X = \{x \mid x \in C_i \text{ and } x \text{ is-a “Cont. Attrib. Evaluation”}\} \] |

It is important to note that our choice of this methodology was determined early on because it allows us to obtain the knowledge of the DM/grid specialist through rules. These rules are then maintained in our compilation of architectural rules, as shown in rules 3 and 4 of table 1. This knowledge can be leveraged not only for process optimization, but also to explain how and when they are applied. It also is important to point out that the entire PSM structure is reused. This enables easy heuristic enrichment, requiring only that the specialist/administrator insert new constraints into the rule base.

5 Experimental Evaluation

In this section, we exemplify how the use of heuristics defined in SMARTBASEG influenced the DM applications performance on the Grid.

For the heuristic evaluation, we generated seven artificial databases with 100k examples and 16 attributes. The first database has only discrete attributes and the second only continuous. The others are composed of continuous and discrete attributes in different proportions. Our intention is to exemplify situations where SMARTBASEG, having knowledge about the application, can dynamically configure and dispatch
tasks to the Grid, influencing the performance of the applications. In figure 5, we see the execution time of three applications, which are different versions of C4.5 algorithm, mining the seven different databases. First, a serial version of the algorithm was executed, in the local machine without use of the grid. Then, the second version of the algorithm was executed on the Grid, with no heuristic. Finally, the third version of the algorithm, using only the constraints and fixes defined in Table 1 and Table 2, was executed on the Grid. For the first and second database, only the task balance heuristic was applied while, for the other databases, the task balance with application knowledge heuristic was preferred.

![Fig. 2. Execution time of three versions of C4.5 algorithm using different approaches](image)

The results show that the automatic configuration based on the constraints improves the performance of the applications. The task balance with knowledge heuristic was the most interesting in terms of performance even if the proportion of discrete and continuous attributes varies. This example shows the major contribution of this work that is the possibility of representing in a knowledge base different heuristics to be used in each specific case. To render these heuristics explicit in the optimization layer enables the programmer of a DM application to be free of concerns related to the application performance on the Grid.

6 Conclusion and Future Work

In this article, we described the manner was implemented in SMARTBASEG a configuration algorithm for preparing a job composed of tasks that represent DM procedures of an application. This was possible because a set of components was supplied to a DM application developer and because both DM applications and grid concepts were characterized in terms of an ontology. This characterization enables the definition of an optimization layer that can decide the best way to submit procedures of a DM application to the Grid. The main contribution of this work is to show that it is possible to provide a dynamic optimization service for better execution of DM applications on the Grid by using knowledge about Grids and mainly about the DM applications being executed. Moreover, this avoid the developer of these applications be concerned with the particularities of the Grid and the means to get better performance on it.

Our research continues to focus on identifying interesting heuristics to be programmed into SMARTBASEG. Specifically, we are investigating the simultaneously
use of multiple heuristics by combining information about the application with dynamic characteristics of the grid. We are also enriching the formalization of the ontology, including more features of DM algorithms and grid environments.

References


Acknowledgements. This work was partially developed in collaboration with HP Brazil P&D.