Future Generation Computer Systems 27 (2011) 703–710

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Active rule learning using decision tree for resource management in Grid computing

Leyli Mohammad Khanli, Farnaz Mahan *, Ayaz Isazadeh

Computer Science Department, Faculty of Mathematical Sciences, University of Tabriz, Tabriz, Iran

ARTICLE INFO

Article history:
Received 11 August 2010
Received in revised form
21 December 2010
Accepted 28 December 2010
Available online 8 January 2011

Keywords:
Grid computing
Rule learning
Decision tree
Resource management
Grid-JQA

ABSTRACT

Grid computing is becoming a mainstream technology for large-scale resource sharing and distributed system integration. One underlying challenge in Grid computing is the resource management. In this paper, active rule learning is considered for resource management in Grid computing. Rule learning is very important for updating rules in an active database system. However, it is also very difficult because of a lack of methodology and support. A decision tree can be used in rule learning to cope with the problems arising in active semantic extraction, termination analysis of the rule set and rule updates. Also our aim in rule learning is to learn new attributes in rules, such as time and load balancing, in regard to instances of a real Grid environment that a decision tree can provide. In our work, a set of decision trees is built in parallel on training data sets based on the original rule set. Each learned decision tree can be reduced to a set of rules and thence conflicting rules can be resolved. Results from cross validation experiments on a data set suggest this approach may be effectively applied for rule learning.

1. Introduction

There are many definitions of Grid computing, with the most commonly quoted definition being given by Ian Foster, "Resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations" [1]. A Grid is a very large-scale network computing system that can scale up to Internet size environment, in which all kinds of computing, storage and data resources, as well as scientific devices or instruments, are distributed across multiple organizations and administrative domains [2,3]. Grid computing is an emerging technology that enables users to share a large number of computing resources distributed over a network. In Grid environments, virtual organizations (VOs) associate heterogeneous users and resource providers such that it is not known how large individual VOs will be, but it is reasonable to imagine resource sharing among populations with tens of thousands of users and thousands of resources [3,4].

Supercomputers, SMPs, clusters, desktop PCs, or even mobile computing devices such as PDAs can be computing resource in the Grid architecture. The Grid resource management system (GRMS) is an important component because it is responsible for storing resource information across the Grid, accepting requests for resources, discovering and scheduling suitable resources that match the requests from the global Grid resources, and executing the requests on scheduled resources. The design and implementation of GRMS is challenging because the Grid is geographically distributed, heterogeneous and autonomous in nature [1,2,4].

Any time that a set of resources needs to be allocated over a set of users we face an NP-complete optimization problem [5]. By managing resources within the active ECA rules we minimize the NP-complete Allocation Problem within On-Line and Off-Line systems.

1.1. The problem

The aim is to find an adaptive resource management related to the real-time Grid environment. The overall problem is to respond quickly and to increase the performance of resource management. For this aim, we need a learning system to update the resource management rules. We need to recognize which rules should be consolidated, which rules should be modified, and which rules should be invalidated. Rule updating, in relation to instances must contain:

1. The ability to change the value of some parameters in the rules.
2. The ability to add new attributes to the original rules and to create new ones.
3. The ability to create new rules that do not conflict with the original rules.

Upon learning our resource management must have the following characteristics:

1. Guarantee to respond to most requests in a real Grid.
2. Respond more quickly with a learning system than without a learning system.
3. Have a higher performance and be more adaptive with a learning system than without a learning system.

* Corresponding author. Tel.: +98 9144111803.
E-mail addresses: mahan@tabrizu.ac.ir, fa_mahan@yahoo.com (F. Mahan).

0167-739X/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.12.016

1.2. Motivation

In reality, resources management can be very complex, and may require co-allocation of different resources, such as specific amounts of CPU hours, system memory, and network bandwidth for data transfer, etc. [3,6]. In this paper we need an active learning database to manage the information and rules. We use Grid-JQA, an architecture supporting such rules in Grid environments, that has been described in previous research [7–9]. Grid Java based Quality of service management by Active database (Grid-JQA) is a framework that provides workflow management for quality of service on different types of resources, including networks, CPUs, and disks. It also encourages Grid customers to specify their quality of service needs based on their actual requirements [7,9].

An active database system (ADBS) is a database system that monitors any situation of interest, and triggers an appropriate response in a timely manner when the interesting events occur [10,7]. The active behavior of a system is generally specified by means of rules that describe the situations to be monitored and the reactions of the system when these situations are encountered. In its general form, a rule consists of a rule event, a condition and an action, known as Event–Condition–Action rules or ECA rules. In Grid-JQA there is an Active Grid Information Server that automatically selects optimal resources using active database ECA rules and requests resource allocation [10,7,11,12].

We focus on active rule learning for improving and learning in resource management in the Grid. Active rule learning is very important for an active database system implementation and we also need the resource management to be updated in Grid computing. But, it is also difficult because many problems arising in rule learning can not be eliminated directly by using traditional database techniques [13,14]. There is a lacks of methodology and support for rule learning. Different representations of concepts may be learned from a set of labeled data, such as neural networks and decision trees [14,15]. A decision tree is useful for representing rules and its learning is reasonably fast and accurate.

1.3. Objective and the claim

Our goal is to have an updated active resource management after learning that is done dependently on new instances in a real event-based environment. Hence, the architecture of our active database has two parts: an Off-Line part and an On-Line part. The On-Line part has static original rules and manages the resource in real time while the Off-Line part performs learning, simulates new examples of the environment, and creates new and updated rules based on the original ones. We focus on 19 ECA rules, as the original rules, that were introduced in [7,11] for resource management in Grid-JQA.

Some rules have static parameters that are determined by an expert, but when these rules are used in a real time environment, these parameters must be changed and updated, based on instances received, in order to improve the resource management performance. Because we receive instances at different times and states, also we must add new attributes such as time and load balancing to the rules. It helps that rule learning be efficient. Our approach to learning different rules is to parallelize the process of learning by using decision trees. In this research, the final representation of the active Grid information server must be ECA rules so we must create rules from decision trees. It is straightforward to reduce a decision tree to rules. The strategy pursued here is to break instances into n partitions based on events, then learn a decision tree on each of the n partitions in parallel [14,16]. A new decision tree will be grown independently when it is needed. At each learning period in each partition, we may have several decision trees related to each rule that must be combined in some way. In [17], the decision trees are combined using metalearning. Also in some partitions, there may be some new decision trees growing independently when there are new attributes in the instances. So the independent decision tree of each rule can be viewed as the agents learning a little about a domain. In our approach, after combining, each decision tree at each partition will be converted to a rule, then the validation of the rules must be checked according to the original rules. Also the performance of a new rule must be evaluated. This new rule set is added to the On-line system and then will be used to respond to new incoming instances.

1.4. Paper outline

This paper is organized as follows: Section 2 demonstrates related work that contains the architecture of the Grid-JQA & AGIS ECA rule engine and decision tree. In Section 3, we describe the active rule learning process for Resource Management in Grid-JQA and we show the use of decision trees for rule learning. In Section 4, we evaluate our approach. Finally, we conclude the paper in Section 5.

2. Related works

Resources management on Grids is a complex procedure involving coordinated resource sharing and responding to the requirements of users. Previous research uses the matchmaking framework implemented in Condor-G. The matchmaking procedure evaluates job and resource rank and requirements expressions to find ideal matches. Additional notable research in this case uses a variety of makespan minimizing heuristics, and the GrADS metascheduler [18]. The GridLab Resource Management System (GRMS) is a job metascheduling and resource management framework. It is based on dynamic resource discovery and selection, mapping and scheduling methodologies. It is also able to deal with resource management challenges [11].

The Globus resource management architecture [7] includes an information service. It plays an important role because it is responsible for providing the information about the current availability and capability of resources, has a co-allocator that is responsible for coordinating the allocation and management of resources at multiple sites, a manager that is responsible for taking RSL (Resource Specification Language) specification, and GRAM (Grid Resource Allocation Management), which is responsible for managing local resources [7].

The Grid-JQA system architecture consists of a Grid portal, Active Grid Information Server (AGIS) (includes ECA resource manager rules and ECA fault manager rules), a fault detector, and GRAM. To execute a job with the Grid-JQA, a user uses RSL to describe a resource type, a resource condition, and the number of resources. RSL is the specification language used by the Globus Toolkit to describe task configuration and service requirements. Then the user sends RSL to a Grid-JQA and the RSL parser extracts the resource type and resource condition and sends them to an AGIS.

The main goal of using Grid-JQA is to provide seamless access to users for submitting jobs to a pool of heterogeneous resources, and at the same time, dynamically to monitor the resource requirements for execution of applications [8,9].

In this paper, we used Grid-JQA, which was introduced in [7] and is represented in Fig. 1 (taken from [7]). The resource manager automatically selects the set of optimal resources among the set of candidate resources using ECA rules and requests resource allocation, thus providing convenience for a user to execute a job. It also guarantees efficient and reliable job execution through a fault tolerance service [7,9]. Initially, the base ECA rules are specified