Robin Hood: An Active Objects Load Balancing Mechanism for Intranet

Javier Bustos Jimenez

Departamento de Ciencias de la Computacion (DCC)
Universidad de Chile.
jbustos@dcc.uchile.cl

Abstract. Scheduling in distributed systems is an important issue, and it has performance impact on parallel processing, load balancing and metacomputing. An optimal load balance in a non-preemptive scheduling enviroment is NP-complete, but preemptive scheduling is polynomial. This paper presents a heuristic for preemptive load balancing based on a multicast channel to communicate the distributed systems, a totally non-centralized architecture and the migration scheme provided by the ProActive tools.

1 Introduction

The scheduling of parallel computations on a set of processors has been an active study area in the fields of paralell computation and load balancing [5]. Theoretical results show that optimal load balance in a non-preemptive scheduling enviroment is NP-complete, but preemptive scheduling is polynomial in time.

Donald McLaughlin says in [5]: "preemptive scheduling in distribuited systems was rare, if not non-existen". Nevertheless, when Active Objects appeared a new research area was (re)opened, by using those objects to migrate jobs from a machine to another one (Virtual or Real machine). Therefore, now one have the tools to make a preemptive scheduler.

1.1 Fundamentals of Load Balance

Load balancing in this paper is a technique to enhance resources, utilizing paralellism exploiting, throughput improvisation, and to cut response time through an appropiate distribution of the application [2]. To minimize the decision time is still one of the objective for load balancing.

There exist two typical load balancing approaches: **Static** and **Dynamic**. Static load balancing is characterized by pre-execution task placement based on a prior knlowledge of the application and target system characteristics (figure 1.a). Dynamic load balancing can adapt to changes in target systems, according to the protocol provided for manage that changes (figure 1.b).

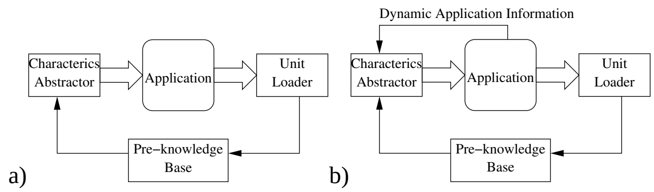

Fig. 1. Static (a) versus Dynamic (b) Load Balancing.

1.2 Contributions

To the best of our knowledge, most of the preemptive scheduling use a centralized architecture (client/server) for the load balancing (see [9, 5, 12, 11]). The contribution of this paper is to present a new totally non-centralized solution: we use a multicast channel to comunicate, and sincronize the processors (following the recomendations of [15]), and using the ProActive [1] tools to migrate jobs between them.

1.3 Related Work

There are several works and research in this area: some use static load balancing like Online Real-Time Schedulers ([9]), Pipeline and Batch Sharing ([6]); and some uses dynamic load balancing like Condor [12], PLRM [11] and CAPE [17].

We focus our research in **dynamic load balancing**. The main differences between our scheme and the related works are: the non-centralized architecture for the load balancer (Condor, PLRM, and others are centralized) and the non-broadcasting of the balance of each node (like CAPE and CONDOR) because that produces an overload of the network.

Other systems, like Amoeba [8] and eCluster [16] are complete micro-kernels that provide load balancer. Therefore, that kind of research are out of the scope of this paper.

1.4 Organization of this paper

The section 2 shows the ProActive principles [1]. The section 3 presents the design of our algorithm (called “Robin Hood”) and finally the conclusions and future of this work in progress are presented.

2 ProActive

ProActive is a Java library (Source code under LGPL licence) for parallel, distributed, and concurrent computing. Also, featuring mobility and security in a uniform framework based on an ideas of Denis Caromel and others [4]. With a reduced set of simple primitives, ProActive provides an API that simplify the programming of applications that are distributed on a LAN, on cluster of workstations, or on Internet Grids [10]. The library is based on an Active Object pattern that encapsulate:

- a remotely accessible object,
- a thread as an asynchronous activity,
- an actor with its own script,
- a server of incoming requests,
- a mobile and potentially secure agent.

ProActive is only made of standard Java classes, and requires no changes to the Java Virtual Machine, no preprocessing or compiler modification. Based on a simple Meta-Object Protocol, the library is itself extensible, making the system open for adaptations and optimizations. ProActive currently uses the RMI Java standard library as a portable transport layer.

For more information please visit [1].

3 Basis of the Robin Hood scheme

The "Robin Hood" scheme has two principles:

1. Every node (processor, JVM, etc.) only know its own load.
2. Job from nodes which have high load (the "rich" ones) are reassigned to nodes with low load (the "poor" ones).

The first principle is easier using `java.lang.Thread` methods to know the CPU load of each node, following the recomendations of [13]. For the second principle, we have to use the `Migration` API of the ProActive libraries [1].

The **algorithm** is:

- If a node note that it is underloaded (less than 60% of the node capacity) then it sends to the multicast channel a message with it owns reference (see figure 2).
- If a node note that it is overloaded (more than 90% of the node capacity) then read from the multicast channel looking for some node underloaded (see figure 3.c). In case that a message arrives from the multicast channel, the node migrate some of its jobs to the underloaded node (figure 3.d).
- Otherwise, nothing is done.

To not overload the network with multicast packages, the node load has to be checked in time intervals, for example: five or ten seconds.

4 The Design of the Robin Hood Load Balancer

For the design of the "Robin Hood" load balancer we use the pattern designs **Singleton** to mantain only one "Robin Hood" per node and **State** to mantain the node states (poor, normal, rich). The UML class diagram is in the figure 4.

But, there was a problem using the ProActive [1] libraries because it use weak migration [7]. Then, only the active thread can make the migration of the active object. For this reason is necesary add to all the objects the method:

Fig. 2. a) Four nodes on equilibrium. b) One node is undeloaded, then sends a Multicast message with its own reference.

Fig. 3. c) A node is overloaded: listen the multicast channel. d) Found an underloaded node, then the “Robin Hood” use `ProActive.migrateTo("A")` over the job representd by a black circle.

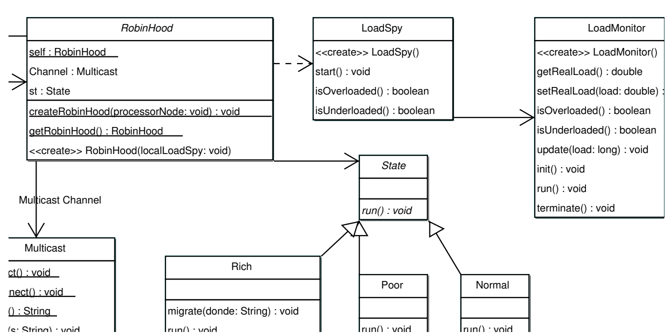

Fig. 4. Robin Hood UML diagram.

```
public void migrate(String URL) {
ProActive.MigrateTo(URL);
};
```

Therefore, if one want to migrate an object `o` the method call is `o.migrate("//host/node")`. By now, one can use tools like *Javassist* [3] to made this change automatically.

5 Conclusions

We present a totally non-centralized load balancer, using the ProActive library for the migration of jobs, and a multicast channel to coordinate the nodes. The next step in our research is to compare "Robin Hood" against the existent ones in dynamic load balancing, for example CONDOR, PLRM and CAPE.

We are currently working on possible solutions for the migration process. Our goal is to make it clear and transparent to the user.

Our future work is to determine the optimal parameters for the load checking, and to improve the migration strategy (choose which job has to be migrated) and the migration itself. Also, we have to work in make the "Robin Hood" a strong fault tolerant [14] tool for load balance in coordination with Christian Delbe of INRIA Sophia Antipolis, France.

References

1. Oasis Group at INRIA Sohpia-Antipolis. "Proactive, the java library for parallel, distribuited, concurrent computing with security and mobility". http://www-sop.inria.fr/oasis/proactive/, 2002.
2. M. Bozyigita. "History-driven dynamic load balancing for recurring applications on network of workstations". *Systems and Software*, 2:61–72, 2000.
3. Shigeru Chiba. "Load-time structural reflection in java". In *In Proc. of ECOOP 2000.*, pages 313–336, 2002.
4. Wilfired Klauser Denis Caromel and Julien Vayssire. "Towards seamless computing and metacomputing in java". *Concurrency Practice and Experience*, 1998.
5. Shantanu Sardesai Donald McLaughlin and Partha Dasgupta. "Preemptive scheduling for distribuited systems". In *Proc. of 11th International Conference on Paralell and Distribuited Computing Systems*, 1998.
6. Douglas Thain et al. "Pipeline and batch sharing in grid workloads". In *Proc. of 12th IEEE Symposium of High Performance Distribuited Computing*, 2003.
7. F.M.T. Brazier et al. "Agent factory: Generative migration of mobile agents in heterogeneous environments". In *In Proc. of the 17th ACM Symposium on Applied Computing*, 2002.
8. Amoeba Group. http://www.cs.vu.nl/pub/amoeba/amoeba.html, 1996.
9. Bhaskar Das Gupta and Michael Palis. "Online real-time preemptive scheduling of jobs with deadlines on multiple machines". *Journal of Scheduling*, Vol. 4:297–312, 2001.
10. Francoise Baude Laurent Baduel and Denis Caromel. "Efficient, flexible, and typed group communications in java". In *Proc. Joint ACM Java Grande - ISCOPE 2002 Conference*, 2002.

11. Yang Xinyu Lu Lina, Liu Longguo. “An agent-based load balancing mechanism: Plrm using java”. In *Proc. of the 37th International Conference on Technology of Object.Oriented Languajes and Systems*, pages 176–181, 2000.
12. Miron Livny Michael Litzkow and Matt Mutka. “Condor - a hunter of idle workstations”. In *Proc. of 8th International Conference on Distribuited Computing Systems*, pages 104–111, 1998.
13. B. Toursel R. Olejnik, A. Bouchi. “An object observation for a java adaptative distributed application platform”. In *In Proc. of International Conference on Parallel Computing in Electrical Engineering (PARALALEC02)*, pages 171–176, 2002.
14. Douglas Thain and Miron Livny. “Error scope on a computational grid: Theory and practice”. In *Proc. of 11th IEEE Symposium of High Performance Distribuited Computing*, 2002.
15. Douglas Thain and Miron Livny. “The ethernet approach to grid computing”. In *Proc. of 12th IEEE Symposium of High Performance Distribuited Computing*, 2003.
16. XGForce. http://www.xgforce.com/eCluster.html, 2002.
17. Saneyasu Yamaguchi and Katsumi Maruyama. “Autonomous load balance system for distributed servers using active objects”. In *In Proc. of 12th International Workshop on Database and Expert Systems Applications (DEXA01).*, pages 167–171, 2001.