computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific Journal of Computing

MIDDLEWARE-BASED LOAD BALANCING FOR COMMUNICATING JAVA OBJECTS

Violeta Felea, Bernard Toursel

LIFL (UMR CNRS 8022) - University of Science and Technology of Lille
59655 Villeneuve d'Ascq CEDEX - FRANCE
Ecole Polytechnique Universitaire de Lille (Polytech'Lille)
{felea, toursel}@lifl.fr

Abstract: *In the context of heterogeneous networks, like clusters of workstations, the design of programming and execution environments aims to automatically adapt execution to fluctuations that may appear in the execution of distributed and parallel Java applications. ADAJ, Adaptive Distributed Applications in Java, addresses this problem, dealing with both parallelism and distribution features. Ease of programming is achieved through an object and method parallelism paradigm. The trade-off between transparency of such a parallelism expression and efficiency is solved by application redeployment, meant to maintain a good performance level. This is the purpose of the load balancing in ADAJ, a dynamic and transparent tool at the middleware level, which exploits information issued from observation of the application, in order to consider both object activity and communication patterns. Communications generate attraction relations between objects and this article presents the evaluation of the load balancing mechanism for a type of asynchronous applications in which the communication aspect is important.*

Keywords: – *dynamic load balancing, load metrics, communication patterns*

1. INTRODUCTION

Heterogeneous systems rise two kinds of problems for distributed applications. The first aspect concerns transparency: applications should be designed as transparently as possible of the heterogeneity of available resources (CPU, memory, operating system). The second aspect concerns execution, which is supposed to support irregularities in the evolution of the application and in availabilities of resources (share of CPU and/or memory with other processes, new provided resources). Java homogenises the heterogeneous platforms through its virtual machine (*JVM*): issues of operating systems, word size or endianness disappear. However, another layer of abstrac-tion, the resource consumption, is missing. Thus, execution cannot be automatically adap-ted to fluctuations which may appear during execution of Java applications.

In the context of distributed execution platforms formed by several Java virtual ma-chines, hosted by a cluster of workstations, efficiency of execution should be achieved not only through conceptual tools (oriented towards distribution or parallelism), but also through a load balancing mechanism.

The dynamic load balancing scheme in ADAJ (Adaptive Distributed Applications in Java) is a transparent tool at the middleware level which makes the execution reactive to irregularities in the evolution of the application and to changes in the resource availabilities. This adaptability requires the use of a monitoring tool, concurrent with the execution of an application, and not prior to it. Monitoring is done using an observation mechanism [1] of the application which allows to predict its evolution in future, depending on the recent past. The ADAJ load balancing strategy uses this kind of information in order to correct, dynamically, detected load imbalances by good initial object distribution or by object redistribution. The JavaParty distributed object model [2] offers the needed features to achieve our objectives: remote creation and migration of remote objects.

Dynamic object migrations are requested when changes in the evolution of the application are detected, with the help of the observation mechanism. Execution is thus made adaptive to dynamic modifications. Similar work is presen-ted in [3], where allocation decisions are made at runtime, depending on the dynamic load measures. Unlike ADAJ, the language defines directives that specify allocation needs for application components.

The focus of this article is on the load balancing strategy which considers only the evolution of the application for good object redistribution, in the context of communicating objects. We present the approach of a dynamic and transparent intra-application load balancing strategy in ADAJ and its evaluation on a concrete application. The article is organised as follows. The next section draws the features of the ADAJ load balancing mechanism and the interactions between its different components. Section 3 makes a short overview of the observation mechanism, while section 4 describes how information issued from the application monitoring is exploited. Section 5 discusses implementation issues concerning the load balancing architecture. Experimental results on a communicating application are given next. Finally, section 7 summarises.

2. LOAD BALANCING SCHEME

In ADAJ, efficiency in the execution of applications is achieved through the use of a profiling tool. This is aimed towards the application behaviour: objects are observed in order to describe their activity during execution. A distributed graph of objects is drawn, dynamically, which reflects both communi-cation links and object activity. The idea of disposing of a graph of communicating objects is not original: dynamic graph partitioning has been proposed for load distribution [4], but in the context of a one-unit graph. Load redistribution based on a distributed graph is a more complex operation, which needs either centralising information, or diffusing it. Both of these solutions are expensive, and consequently, in ADAJ, no optimal solution is searched, but an improvement of object distribution, in respect of load balance and communicating objects locality (remote objects communicating intensively should be brought close together).

The observation information is exploited by the three components of the load balancing mechanism:

- the observation component,
- the decision component,
- the correction component.

The observation component contains a load extractor and the object relation observer. The first defines the load of a virtual machine, depending on the observation information (offered by the second) and the number of threads. Every machine load is communicated periodically to the decision component which analyses it and determines the machines which are participating in the load redistribution. The machines concerned are notified and they apply, in a distributed manner, the correction algo-rithm, which achieves the redistribution process.

3. OBSERVATION MECHANISM OF RELATIONS

Remote objects are particularly interesting in load balancing mechanisms because they are able to migrate. In ADAJ, the load balancing mechanism is based on an observation tool [1] of the evolution of the application. The observation concerns only the remote objects, as they can balance load by migration. The remote objects which are to be observed are called *global objects*. The Java standard objects, called *local objects*, are not remote accessible, cannot migrate and are not observed.

The ADAJ observation mechanism of relations maintains a history of the relations of every global object with the environment. We distinguish three types of relations:

- of a global object with another global object (on the same, or different virtual machines),
- of a global object with all other local objects,
- of all other global or local objects with a particular global object.

In object-oriented environments, these relations are generated by method invocations. This remark allowed us to quantify relation intensity by the number of method invocations and not by the method execution time or parameter size as in other projects (Dome [5] or Isatis [6]). Method invocations generate work on global or local objects, and work can only be created through method invocations.

For each of these relations, counters are associated. Counter values are submitted to a smoothing mechanism in order to take into account both past evolution and present value. Smoothing is required because the current behaviour should be weighted with the previous ones, since sudden, not persistent fluctuations are neglected.

This ingenious idea of relation quantification is less costly and less complex than other techniques and gives a rating of object activity.

4. EXPLOITING OBSERVATION INFORMATION

Load Extractor. Load in a Java virtual machine is generated by the activity of the objects which it contains. Methods invoked on objects, generating the activity, are executed in the main thread of the virtual machine, or in the user threads. Thus, the load of a virtual machine is generally measured by the number of threads. In fact, in the *JVM* there is only one thread running (on a mono-processor machine), the other threads being runnable or not runnable. The portable information which can be extracted, from the virtual machine, is the number of active threads, which can be either runnable or not

runnable. The difference between the two kinds of threads is done in ADAJ using the *workload of JVM*s, the two criteria defining the load of a *JVM*. The workload of a *JVM*, noted *WP*, sums the workloads of every global object the *JVM* contains. As mentioned before, an object's workload is generated by method invocations. All input invocations and also invocations towards local objects characterise the workload of a global object.

Decision Component. The decision component classifies *JVM*s as overloaded, normally loaded and underloaded. The algorithm is the follo-wing:

- if there are a lot of threads,
  - if *WP* is close to zero (not runnable threads), then *JVM* is underloaded,
  - if *WP* is important (running threads), then *JVM* is overloaded,
- if there are few threads (no work in the *JVM*), then *JVM* is underloaded,
- if there is a normal number of threads,
  - if *WP* is close to zero (not runnable threads), then *JVM* is underloaded.

Close to zero and important values are defined using a customised K-Means algorithm: in a preliminary phase, detection of their existence is done (using statistical metrics, as variation coefficient), and then, if it is the case, the values having these properties are identified, using the K-Means method. This identification forms three classes of values: the class of values close to the smallest value (associated to the close to zero values), the class of values close to the biggest value (associated to the important values), and the class of values around the mean (associated to the normal values). These associations are possible because "close to zero"/"few" and "important"/"a lot" characteri-sations (for the *JVM* workload and respectively, for the number of threads) are relative, and not absolute.

Correction Component. The correction component concerns only the overloaded machines, which decide on the objects to remove and on their destination.

Between all global objects a virtual machine has, some are particularly interesting to remove: objects which do not have strong communi-cation links towards the other global objects remaining on the machine, and those which have an average workload. The first feature avoids the generation of new remote communi-cations, while the second assures some work-load will be really removed.

The two constraints are simultaneously considered using an aggregation function, the weighted sum [7]. This technique imposes that values were on a same scale, that's why relative values are considered.

The best classified object, in the respect of the previous function, is to be moved to another virtual machine. This decision depends on its external attraction (communication with global objects in another address space), and on the workload of the destination machine. The previous technique of aggregation is used in order to take into account both criteria.

5. TECHNICAL ISSUES

The three components of the load balancing mechanism (see figure 1) are instances of the: *LocalLoad* remote class (for the load extractor), *LoadDecision* local class (for the decision component) and *LoadBal* remote class (for the correction component).

The *LocalLoad* object is responsible of extracting load measures, from both the observation mechanism (implemented by a local *AsynchObserver* object, which is updated by a remote *LocalJVMObserver* object), and the execution environment of the Java Virtual Machine (number of threads). The two measures are packed in a local *InfoLoad* object, and transmitted to the decision component.

The *LoadDecision* object is unique and has the functionality of applying the decision algorithm for the load measures gathered from all machines. It activates, if necessary, the *LoadBal* object on every overloaded machine, which applies the location policy, implemented by a local *Decision* object. Remote observation information is recovered from the remote *LocalJVMObserver* objects of the underloaded machines.

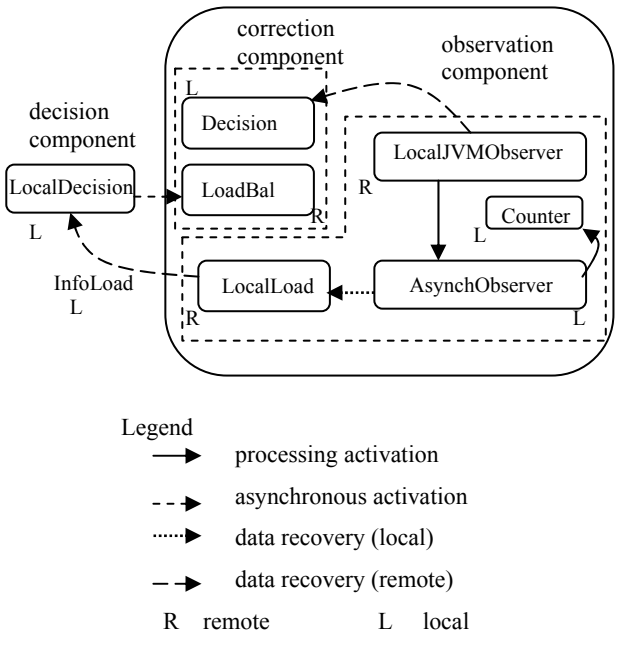

Fig.1 – Components of the ADAJ load balancing scheme.

6. EXPERIMENTAL RESULTS

The load balancing mechanism in ADAJ was experimentally tested for a distributed and parallel genetic algorithm solving the TSP (Travelling Salesman Problem) problem. This algorithm uses the island model, where the initial population is divided into sub-populations, on which a classical genetic algorithm is applied, in parallel, for a number of evolutions. Afterwards, the best individuals are exchanged between sub-populations. A possible implementation in ADAJ makes the sub-populations remote objects that can migrate.

Experiments consider the execution of the algorithm on a cluster of workstations, with no external load. Initial distribution of objects is voluntarily unequal. Previous tests [8] show good behaviour: gains up to 30% in execution times are achieved, compared to a version which does not use the load balancing mechanism. The tendency is to approach to an equal distribution of objects on every node, which minimises considerably the waiting time in synchronisations.

The TSP application shows situations where the quantity of processing to be executed is not necessarily well distributed, because of unequal sub-population distributions or because of diffe-rent sub-population sizes. But communications are not taken into consideration. The focus of this article is on another kind of application, which makes use of communication. It is the simulation of an algorithm which solves numerical problems, using an iterative wave-form method (see the Medico Akzo Nobel problem [9]). The direct method finds the exact solution after a finite number of operations, while the iterative method gives an approxi-mative solution, after a number of iterations, but has a lower complexity.

The skeleton of this kind of algorithm consists of local computing phases, in parallel over different data, and of information exchange with the neighbour, in a ring fashion.

In ADAJ, the simulation of such an algo-rithm considers a number of global objects, every global object executing a sequence of communication towards the next global object, followed by a local computation. The comm-unication consists of requesting the same kind of computation, as the local one.

Internal towards External Communication In an object-oriented application, communication links are generated by method invocations. Communication between global objects placed on different virtual machines includes seriali-sation and network passing cost. When global objects are located on the same virtual machine, the network cost is eliminated. The difference between the two kinds of communication induces the definitions of *internal communi-cation*, corresponding to method invocations between global objects in the same address space, and of *external communication*, which is generated by method invocations between global objects in different address spaces.

The objective of the load balancing mecha-nism is to take into account the communication links in the correction algorithm in order to avoid creating new external communications, and to replace external communications with internal ones. Even if an internal communi-cation is always remote (using the serialisation mechanism), the execution time can be im-proved eliminating the network traffic overhead.

This results from a test which makes n invocations between two remote objects. The execution platform was made of two homogeneous PIII machines, 733 MHz, having 128M RAM, linked by 100 Mb/s throughput. Table 1 shows an average slowdown of 22% for an external communication, compared to an internal one[1].

Table 1. Execution times and overheads of external communications compared to internal ones

	internal comm (ms)	external comm (ms)	overhead (%)
n=500	117	140	19.65
n=1000	224.66	275.33	22.55
n=5000	1094	1366	24.86
n=10000	2217.66	2728.66	23.04

The network throughput has an important role in these measurements: a slow traffic makes external communication even slower.

The penalty induced by the external com-munication is even more important if communication optimisation can be introduced: remote objects which are on the same virtual machine can communicate locally (see the Java-Party 0.98 version). In this case, a remote com-munication can become an internal local one, using object migration.

Communicating Application The communi-cation pattern presented in figure 2 shows that every object executes, concurrently, a compu-tation requested by the previous object, through a communication, and a local computation (ex-cept for the last one which does not perform communication, and for the first one which is not requested the computation). The local pro-cessing is blocked during the communication, because of the synchronous call.

[1]values are averages of 5 execution times, from which the best and the worst time were removed

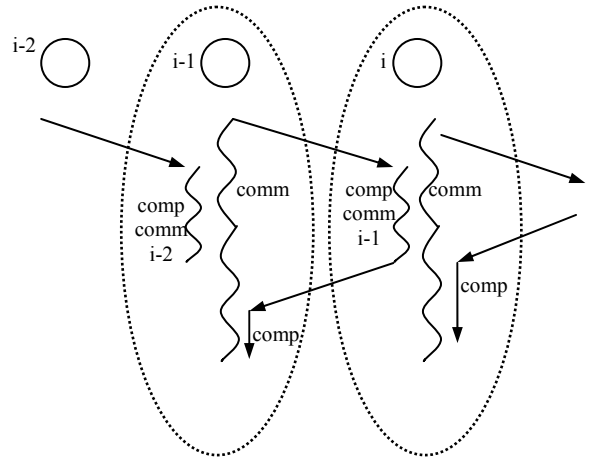

Fig. 2 – Skeleton of a communicating application.

Communications are considered by the correction component. The load balancing mechanism in ADAJ balances the load, targeted towards communication optimisation, and does not react to communication imbalances. Thus, the initial distribution of objects is voluntarily unbalanced, and makes communications random. For example, if objects are indexed from 0 to 12, their initial distribution on four machines is shown in figure 3.

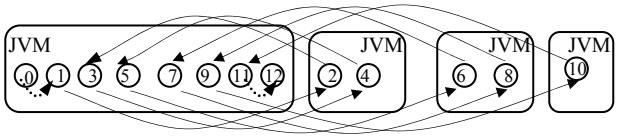

Fig. 3 – Initial deployment of the communicating application.

This distribution generates 10 external communications, and 2 internal ones. The load balancing mechanism should react to the load imbalance, correct object distribution, by a good placement, which consequently increases the number of internal communications.

Application Behaviour The objective of the experiments on the communicating application was to analyse the decisions of the load balancing mechanism in respect of communi-cations. Three cases were tested, concerning the choice of the destination machine:

- both communication links and machine workload are important,
- only communication links are important,
- only workload of destination machines is important.

In the first situation, two of the final distributions in figure 4, show a good load balance, and external communications were diminished from 10 to 6 (in average).

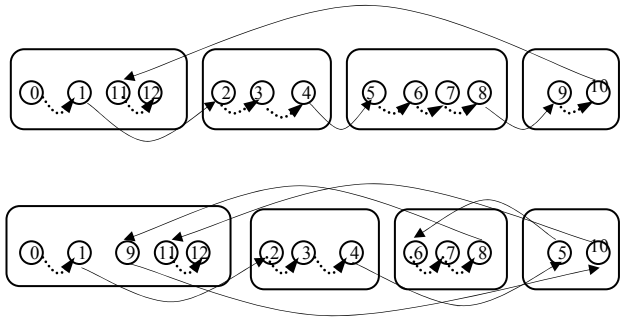

Fig. 4 – Final deployments of the communicating application.

If communications, only, are considered, experiments show a similar behaviour, because the destination machines are however chosen between the least loaded. On the contrary, when communications are not at all considered, in the third case, loads are balanced, but there is no improvement for external communications (figure 5), or worse, they may be increased.

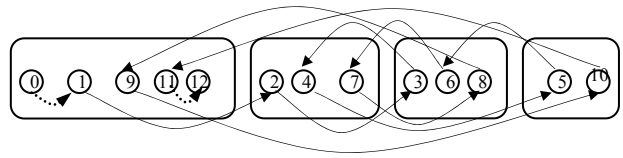

Fig. 5 – Final deployment of communicating application.

These experiments testify that load balancing in ADAJ is able not only to balance loads, but also to reestablish a good communication pattern, lost because of inadequate object deployment.

7. CONCLUSION

Efficiency of execution for distributed and parallel object-oriented applications is an important issue in designing execution environments. In this article we have presented an approach to deploy transparently and dyna-mically applications over a cluster of work-stations. Our solution is a load balancing mechanism at the middleware level which uses profiling information on the application behaviour, in terms of object activity and communication links. The main hypothesis is that application behaviour in the near future re-sembles to its behaviour in the recent past.

Our previous results [8] showed good performances, compared to executions using no load

balancing mechanism strategy. Gains up to 30% in execution times were measured. This article is focused on the behaviour of a com-municating application, when load balancing is activated. Random redistribution of objects may balance load, but they cannot always improve communication links, by bringing closer (on the same Java virtual machine) remote communi-cating objects. The load balancing mechanism in ADAJ considers communication links during the correction policy and thus is able to recover a good communication pattern, by improving the number of internal communications.

8. REFERENCES

[1] A. Bouchi, R. Olejnik, and B. Toursel. A New Estimation Method for Distributed Java Object Activity. In IPDPS 2002 - Workshop on Java for Parallel and Distributed Computing, Fort Lauderdale, USA, 2002.

[2] M. Phillippsen and M. Zenger. JavaParty - Transparent Remote Objects in Java. In ACM 1997 Workshop on Java for Science and Engineering Computation, Las Vegas, USA, June 1997.

[3] A. Corradi, L. Leonardi, and F. Zambonelli. High-Level Directives to Drive the Allocation of Parallel Object-Oriented Applications. In Proceedings of HIPS'97, Amsterdam, Pays Bas, 1997.

[4] K. Schloegel, G. Karypis, and V. Kumar. Graph Partititioning for High Performance Scientific Simulations. Technical Report: TR 00-018, Dept. of Computer Science and Engineering, University of Minnesota, 2000. To be included in CRPC Parallel Computing Handbook.

[5] J. Arabe, A. Beguelin, B Lowekamp, E. Seligman, M. Starkey, and P. Stephan. Dome: Parallel programming in a heterogenous multi-user environment. Technical report, Carnegie Mellon University, Avril 1995.

[6] M. Banâtre, Y. Belhamissi, V. Issarny, I. Puaut, and J.P. Routeau. Adaptive Placement of Method Executions within a Customizable
Distributed Object-Based Runtime System – Design, Implementation, and Performance. ISSN 1350-2042 TR 64, IRISA and CRIN-Nancy, 1994.

[7] P. C Fishburn. A survey of multiattribute/ multicriteria evaluation theories. In S. Zionts, editor, Multicriteria problem solving, pages 181–224. Springer Verlag, Berlin, 1978.

[8] V. Felea. Exploiting Runtime Information in Load Balancing Strategies. In P. Kacsuk and D. Kranzlmüller and Z. Németh and J. Volkert, editor, Distributed and Parallel Systems – Cluster and Grid Computing, pages 21–29, Linz, Austria, 2002. Kluwer Academic Publishers.

[9] CWI - The National Research Institute for Mathematics and Computer Science in the Netherlands. Test Set for IVP Solvers. http://www.cwi.nl/ftp/IVPtestset/descrip.htm.

Bernard Toursel *is Professor at the University of Science and Technology of Lille, France. He was vice-director of the Ecole Universitaire d'Ingénieurs de Lille (nowadays Polytech'Lille), then director of the Laboratory of Computer Science (LIFL), UMR CNRS 8022. Today he is Vice-President in charge of Information and Communication Technologies. His scientific research concerns the field of parallel and distributed systems and processing.*

Violeta Felea *received her B.A. degree in computer science from "Al. I. Cuza" University, Iasi, Romania, in 1998 and the M.S. and Ph.D. degrees in the same field at the University of Science and Technology of Lille, France, in 1999 and 2003, respectively. Her dissertation is a study of methodologies for the design of parallel and distributed Java applications and of tools for the efficiency of execution. She joined the PALOMA team at the Laboratory of Computer Science of Lille, in 1998, and today she is ATER (Research Assistant) at the Polytech'Lille. Her research interests include distributed and parallel object-oriented programming, scheduling algorithms, tools for object migration.*