Integration of Load Balancing in CORBA Environment

Jaka Močnik[1], *Roman Trobec*[1], *Borut Robič*[2]

[1]Jožef Stefan Institute
Jamova 39, Ljubljana, Slovenia
[2] University of Ljubljana
Faculty of Computer and Information Science
Tržaška 25, Ljubljana, Slovenia
e-mail: `jaka.mocnik@ijs.si`

Abstract

A framework for distribution of program objects in heterogeneous computer systems using CORBA is described. An object was implemented for collecting information on actual load of available workstations. These results are then used as the decision criteria for load balancing of the system. The implementation is tested on an algorithm performing an exhaustive search over all topologies with all possible paths between nodes in a network of n connected nodes in order to find the topology yielding minimal sum of average and maximal distance between all pairs of nodes.

1 Introduction

CORBA [1] applications are usually complex applications consisting of a large set of communicating objects performing different operations. Each object is implemented by a server, usually in the form of an operating system process. CORBA provides for location- and implementation-transparent access to objects, therefore such an application can easily take advantage of a network of computers while hiding hardware and operating system specifics of network computers from the programmer..

In order to enable a CORBA-based application to utilize such a network as good as possible, the objects required by an application should be distributed among the computers on the network. Since the network configuration and computer properties are usually not known in advance, manual distribution of objects leads to either very complex solutions, distracting the programmer from the actual problem, or an overly simplified one, resulting in poor utilization of the network resources.

The aim of this paper is to show a practical example of a framework for load distribution in a CORBA environment that is easy to use and can take advantage of existing object implementations. Section 2 discusses some general properties of load distribution and framework without a specific implementation in mind. Section 3 presents an implementation example using a freely available CORBA 2.2 compliant Object Request Broker (ORB). Section 4 shows the performance of the proposed framework with a test application and the test results with some simple load distribution policies. Section 5 gives the conclusions and summarizes the work.

2 A Model for A Load Distribution Framework

2.1 Load Distribution

Load distribution is a process of assigning entities to targets in such a way that certain performance criteria, like cumulative load on all machines or the time needed for performing a task, are optimized.

Load distribution can be solved either statically or dynamically:

a) *static load distribution* assigns entities to targets upon startup of the application. This approach is suboptimal in most cases, as resource requirements of an entity or system resources provided vary. For example, the complexity of a certain operation might depend on the results of a previous operation. A typical scenario where the system resources provided by a target change dynamically is the execution of a distributed application in a multi-user environment, where each workstation processes requests from a number of users and therefore does not provide constant computing power for our application.

b) *dynamic load distribution* assigns entities to targets on-demand and according to the actual load situation. This approach is obviously better but requires less strict semantics of the application in order to apply load distribution mechanisms.

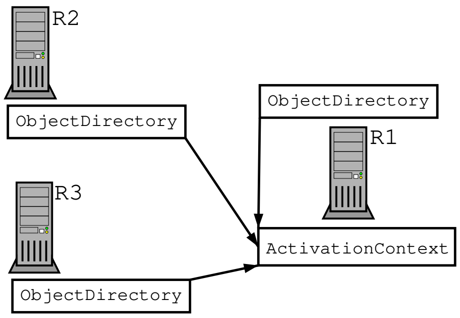

Figure 1: Structure of OAF, the implementation repository used in the proposed load distribution framework

2.2 Load Distribution in a CORBA Environment

In the context of CORBA object model, the application consists of a series of operations invoked on objects that implement the interface containing the required operation. Such objects are usually implemented as server processes that may run on any machine on the network that has the executable installed. The most natural load distribution model for a CORBA environment would be assignment of object instances (running server processes) to computers on the network.

A general model for load distribution in a CORBA environment as described in [5] represents three levels: the CORBA Object level, the object instance level and the system level. As stated above, the proposed framework handles load distribution on the system level. It assigns processes and, indirectly, CORBA objects and operation invocations to machines on the network.

Such an assignment strategy can be either sender-initiated, receiver-initiated or mixed [6]. The chosen strategy was sender-initiated where the source of workload (a client) looks for an appropriate destination (a server) that would process the workload. This strategy can be implemented in the CORBA object model in a straightforward way. The client that wants some operations invoked on a certain object, requests a reference to an object implementing the desired interface. The load distribution strategy selects one from all the available implementations and returns a reference to it. The application then uses the reference to invoke operations on the object.

3 Implementation of a Load Distribution Framework

The load distribution framework has been implemented using ORBit, a freely available CORBA 2.2 compliant ORB, and Object Activation Framework (OAF, figure 1), an implementation repository. An implementation repository is required as CORBA itself does not provide any mechanism for locating and executing programs that serve as object implementation.

OAF, the Object Activation Framework, implements two interfaces: ActivationContext and ObjectDirectory. The latter provides a list of all object implementations for a particular machine. The former lists all known ObjectDirectories and provides the application with a context activation of objects that match programmer's requirements, specified by a simple query language. The common requirement that the programmer has to specify is the interface that has to be implemented by the object. An example of object activation in pseudo-code is:

```
obj_ref = OAF::ActivationContext::activate(requirements, sel_order, flags);
```

When multiple objects match the requirements, OAF selects the first one, runs the executable if necessary and returns the object reference to the application.

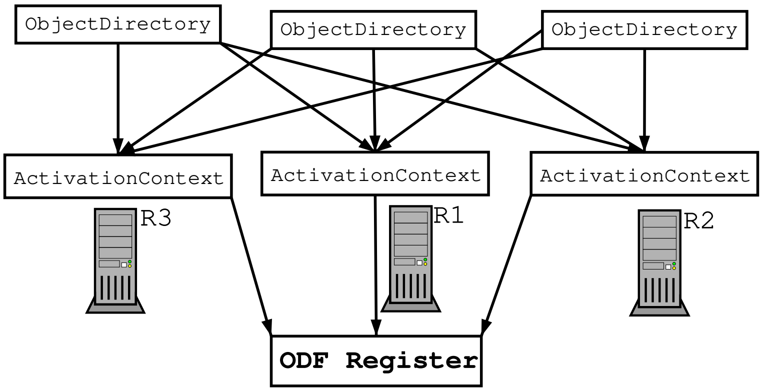

Figure 2: Organization of ODF, the proposed load distribution model

The proposed Object Distribution Framework, ODF - shown in figure 2, consists of the ODF::Register interface that initially reads a list of all computers and, using remote or secure shell UNIX commands, activates a local ActivationContext and ObjectDirectory. When this bootstrapping process is finished, all known ObjectDirectories are added to each ActivationContext.

If an application using ODF is started on any of these machines it first retrieves a reference to the ODF::Register object that runs centrally, on a single machine on the network. Register supplies the reference to the local ActivationContext which in turn lists all object implementations known to OAF on any of the machines on the network. When the application requests a reference to an object, the ActivationContext will be queried for a list of all the object implementations matching the specified requirements, and the ODF will select the most appropriate one based on the implemented load distribution policy and return a reference to it. The application can then use this reference to invoke operations on the object. In this way, the distribution of objects is completely hidden from the programmer, as object activation as described above is simply changed to:

```
obj_ref = ODF::Register::activate(requirements, sel_order, flags)
```

Furthermore, it is not necessary that the object implementations know anything about ODF except if they want to use load distribution strategies themselves. The binary compatibility of the application code is therefore guaranteed and development time can be greatly reduced by code reuse.

The only changes that have to be made to the code is to use CORBA's Dynamic Invocation Interface (DII) for invoking operations on objects instead of static invocation through stubs. Only DII allows for deferred synchronous operations so that the result of an operation can be collected only when required. If static invocation through stubs is used the execution blocks until the result arrives, which is obviously inappropriate for parallel programming. ODF also provides convenience functions that simplify the use of DII both when invoking operations and waiting for the results.

Besides ActivationContexts and ObjectDirectories of all network computers, ODF also keeps other information about the machines. This information which includes the current load of the machine and its relative speed index may be used by the load distribution algorithm to select the target machine in a more efficient manner. The current load is reported by objects implementing the Overseer::CPU interface, which are activated on each computer. This interface currently works on HP-UX, Solaris and Linux operating systems.

Name	Architecture	Operating System
R1	Intel PIII 450 MHz	Linux 2.2.16
R2	Intel Celeron 466 MHz	Linux 2.2.14
R3	HP PA-RISC	HP-UX 10.20
R4	Sun 4U SPARC Ultra-30	SunOS 5.6

Table 1: Network configuration

R1	R2	R3	R4
0.352	0.288	0.184	0.176

Table 2: Computer weights

4 Test results

The test application was a program that performs an exhaustive search over all possible configurations of links between nodes in a given X by Y mesh of degree D. The goal is to find the link configuration yielding minimal sum of average and maximum distance between any two nodes. The problem can be simply parallelized so that the set of all possible link configurations is partitioned into equally sized subsets and each of these subsets is then inspected for its local optimum.

To create such an application an interface Test::MeshAnalyzer was implemented. An object implementing this interface is given the mesh parameters and a set of link configurations as a parameter. It can find optimal link configurations within this set with operation*find_optimal*.

A server implementing this interface was then compiled and installed on four different workstations forming our heterogeneous environment, presented in table 1:

The application forms the next subset of link configurations, uses ODF to activate a Test::MeshAnalyzer object and invokes its *find_optimal* operation, passing it this subset as the parameter. This is repeated until all the subsets have been sent to the objects that will process them.

Four basic load distribution policies have been tested: random distribution, round-robin distribution, weighted distribution and the load threshold distribution. In the weighted algorithm, the probability of selecting a machine is related to its weight. The weights from table 2 were set by measuring the time of processing a small subset of link configurations. The load threshold algorithm measures the load of a machine and compares it with a specified threshold and selects the computer if the load is beneath the threshold. If all machines are overloaded, the algorithm waits periodically checking load indices until at least one falls under the threshold.

All tests were performed on a 32 by 32 mesh of degree 10. The size of the link configuration subset analyzed at a time was 5000 configurations. The solution yields 8 optimal link configurations with an average distance between nodes 4.055 and maximum distance 6.000.

The results of the test, presented in table 3, show that the *weighted* algorithm achieves maximum performance with the *load threshold* algorithm close behind it. This was the result of a larger overhead involved because of load checking. The *load threshold* algorithm was expected to outperform the *weighted* because of its dynamic nature. To prove this, the test should span over a longer period and involve real third-party load imposed on the test machines. *Round-robin* algorithm does not perform that well as it does not take into account any information on the available computers. The *random* algorithm was expected to achieve similar results as *round-robin* as the number of activations performed during the test is very large. Actually it performed even worse, as it often chose the slower computers.

	R1	R2	R3	R4	Total time used for solution
Random	684 s	610 s	1992 s	2066 s	2066 s
Round-robin	906 s	1027 s	1516 s	1579 s	1579 s
Weighted	1013 s	1249 s	1041 s	1208 s	1249 s
Load Threshold	1336 s	1053 s	959 s	1224 s	1336 s

Table 3: Time used by testing application using different load balancing algorithms

5 Conclusions

It has been shown that in an heterogeneous network environment the proposed load balancing scheme allows for better utilization of network resources and better performance of CORBA based applications. Minimal changes to existing applications are necessary to use the load balancing framework. The load distribution integrates seamlessly into CORBA infrastructure of ORBit and OAF. This results in maximum possible reuse of existing object implementations, thus reducing development time.

References

[1] Object Management Group, The Common Object Request Broker: Architecture and Specification - Revision 2.2, *ftp://ftp.omg.org/pub/docs/formal/98-07-01.ps*, 1998

[2] Object Management Group, Discussion of the Object Management Architecture, *http://www.omg.org/library/oma/oma-all.ps*, 1997

[3] Vinoski, S., CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, in IEEE Communications Magazine, Vol. 35, No.2, February 1997

[4] Schnenkenburger T., Rackl G., Implementing Dynamic Load Distribution Strategies with Orbix, in International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '97), Las Vegas, Nevada, Volume II, p. 996-1005, CSREA, ISBN 0-9648666-6-8, 1997

[5] Schnekenburger T., Automatic Load Distribution for CORBA Applications, Component Users Conference, Munich, Germany, SIGS, ISBN 1-884842-57-7, 1996

[6] Feng, M. D. and Yuen, C. K., Dynamic Load Balancing on a Distributed System, in Proceeding of the 6th IEEE Symposium on Distributed Processing, p. 318-325, Dallas, Texas, 1994

[7] Barth, T., Flender G., Freisleben B., Thilo F., Load Distribution in a CORBA Environment, in Proc. 1999 International Symposium on Distributed Objects and Applications (DOA'99), p. 158-166, Edinburgh, Scotland, September 1999.