Quick Clicks

Conference Chair Message

Program Co-Chairs Message

Table of Contents

Proceedings of the ISCA
16th International Conference

PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS

Reno, Nevada USA
August 13-15, 2003

Editors: S-M. Yoo and H. Y. Youn

A Publication of
The International Society for
Computers and Their Applications - ISCA

ISBN: 1-880843-48-X

On the Development of an Enhanced Least-Loaded Strategy for the CORBA Load Balancing and Monitoring Service

N. Arapé
Applied Computing Institute
University of Zulia
Maracaibo, Zulia, 4001, Venezuela
narape@ica.luz.ve

J. A. Colmenares
Applied Computing Institute
University of Zulia
Maracaibo, Zulia, 4001, Venezuela
juancol@ica.luz.ve

N. V. Queipo
Applied Computing Institute
University of Zulia
Maracaibo, Zulia, 4001, Venezuela
nqueipo@ica.luz.ve

Abstract

The least-loaded strategy (LL) is a built-in dynamic algorithm for the proposed standard CORBA Load Balancing and Monitoring Service. This paper describes this algorithm and identifies three improvement opportunities. We propose a strategy, called enhanced least-loaded (eLL), that incorporates the corresponding improvements. Simulation results demonstrate that eLL has potentially better average performance than LL.

Keywords: Load balancing, CORBA-based technologies, least-loaded strategy.

1 Introduction

A new generation of large-scale enterprise information systems is increasingly available (e.g. ERP, BPM, e-commerce and financial systems). Typically, these are object-oriented distributed systems characterized by high reliability, performance, scalability, availability and serviceability. A successful technique to improve the reliability, performance and scalability of these systems is to balance the processing load among multiple server hosts [2, 17, 14].

CORBA [11] is an object-oriented distributed-system architecture developed by the Object Management Group (OMG). Characteristics such as multi-platform support, programming language independence, highly standardization, specification stability and maturity make CORBA specially suitable for setting the foundation of large-scale systems. Therefore, its popularity as computing middleware is rising.

There have been important efforts to extend CORBA functionality to support load balancing and monitoring; see, for example, TAO [13, 12], VisiBroker [9], Orbix [7] and MICO [16]. However, these initiatives have often resulted in proprietary mechanisms designed for specific platforms and applications, so they cannot be extended to heterogeneous environments and new applications. To consolidate the previous efforts and overcome this problem, the OMG issued a Request For Proposal [10] to develop a standard for load balancing and monitoring for CORBA-based applications. The corresponding Revised Joint Submission [8] can be considered the most relevant work in this subject and OMG has recommended its adoption.

The cited submission includes three load-balancing strategies that compliant implementations must support: two static strategies (round-robin and random) plus a dynamic strategy (least-loaded). Dynamic strategies potentially outperform static ones because they consider the current system-state information to make load distribution decisions [14]. These strategies are specially suitable when unbalances result of fluctuations on the rate of arriving requests or a high variability of the request execution time.

Specifically, the least-loaded strategy (LL) was initially proposed by IONA [6] with TAO and PrimsTech also offering implementations [3, 15]. In our experience, LL is an effective and easy to implement load

balancing algorithm with a reasonable average performance and little overhead. However, it makes no consideration about heterogeneous multicomputer systems (a very frequent scenario) and lacks a mechanism to detect inactive and/or slightly loaded nodes. Consequently, some nodes (i.e. faster nodes) are usually idle whereas other nodes (i.e. slower nodes) are considerably loaded, so LL does not take full advantage of the processing capacity of the nodes.

Additionally, although the proposed specification concentrates predominantly on the push style load monitoring because of pull style scalability issues, current CORBA load balancing implementations generally only incorporates the periodic pull style load monitoring due to its simplicity. The main shortcoming of the periodic pull style is that the user must properly set the pull period. This is not an easy task considering that long periods could make the strategy work with non-updated load information and short periods could cause network congestion. On the other hand, push style has two weaknesses: i) the load information could risk being non-updated if during a long period of time the load value does not cross any threshold, and ii) push style requires more elaborated load monitors.

This paper discusses an improved version of LL, called enhanced least-loaded (eLL), that overcomes the aforementioned problems. The eLL combines the push and pull load-monitoring styles, and includes two additional thresholds (*idle* and *busy*) and a processing capacity compensation factor.

This paper is organized as follows. Section 2 describes the CORBA least-loaded strategy. Improvement opportunities and the corresponding modifications are explained in Section 3. Section 4 presents the simulation cases and describes the methodology employed to compare the performance of the load balancing algorithms. Section 5 discusses the results of the performance evaluations. Finally, Section 6 states the conclusions and presents upcoming research.

2 CORBA Least-loaded Strategy

The least-loaded strategy (LL) is a dynamic non-preemptive load balancing algorithm based on the Graham's List Processing Algorithm [4] that assigns each job to the currently least loaded group member[1]. This section describes LL in terms of inherent policies [17, 14] to the behavior of dynamic load balancing algorithms. Many aspects of these policies are direct consequences of the Load Balancing and Monitoring Service design [8].

2.1 Information Policy

LL follows the centralized approach, therefore all information required to distribute the work between the nodes is concentrated at a single location (the load manager). The load manager gathers the load information through the load monitors located at each node. The information collecting can occur in two styles: i) pull: load manager requests the loads to load monitors, and ii) push: load monitors send load reports to the load manager. The proposed specification concentrates predominantly on the push style due to scalability issues introduced with pull style [8]. In addition, this specification does not specify whether the load information actualization must be periodic, state-change (threshold) driven, or a combination of both, thus we can assume these types of information policies are valid. For example, TAO's preliminary implementation of the Load Balancing and Monitoring Service and least-loaded strategy periodically collects the load information using the pull style.

Load monitors report raw load values (load indexes) of the corresponding group members. The load manager (specifically, the load analyzer) converts the raw load values into effective load values and uses them to make balancing decisions. The conversion rule is given as follows [2]:

$$el_k = \left\lfloor \frac{d \cdot (el_{k-1} \cdot tol + r_{k-1} \cdot pbl) + (1 - d) \cdot nrl}{tol} \right\rfloor \quad (1)$$

where:

el_k: is the current effective load value (notice that it is determined using integer division).

el_{k-1}: is the previous effective load value.

d (dampening): determines how raw load values change with each update. Its purpose is to avoid the algorithm overacts in presence of load peaks. The dampening value must be in the interval $[0.0 \ldots 1.0)$ (default=0).

tol (tolerance): allows load values to be arranged into equivalence classes and its value must be non-zero (default=1).

[1] According to the Revised Joint Submission [8] a *group* is a collection of CORBA objects of the same type which are balanced by the Load Balancing and Monitoring Service, and these objects are called *group members*.

[2] The Revised Joint Submission of CORBA Load Balancing and Monitoring Service does not clearly describe how the effective load value must be calculated.

r_{k-1} (receiver): only takes two values: 0.0 and 1.0. $r_{k-1} = 0.0$ when the group member was not the receiver in the previous load transfer; otherwise, $r_{k-1} = 1.0$.

pbl (per-balance-load): represents an estimation of the additional amount of work that the selected location must do as a consequence of each time the balancer makes a load-balancing decision and location-forwards a client (default=0).

nrl: is the new raw load value reported by the load monitor.

2.2 Transfer policy

LL uses two thresholds to determine whether a group member is in a suitable state to participate in a load transfer. These are:

1. *reject-threshold*: if the effective load value of a group member is greater than this threshold, the strategy does not consider this member as a potential receiver. If all members of a group have an effective load greater than *reject-threshold*, the load manager sends a notification to the client (raises a `TRANSIENT` exception) indicating that the system temporarily is not able to satisfy its request.
2. *critical-threshold*: if a group member has an effective load value greater than this threshold, the load manager instructs the member to redirect the next request back to him. This member is then removed from the potential receiver list.

If *reject-threshold* and *critical-threshold* are both non-zero, *critical-threshold* must not be smaller than *reject-threshold*.

The preceding description reveals that LL follows the sender-initiated approach [14, 5] because the overloaded members initiate the load transfers. On the other hand, the load manager always participates in load transfers. The load manager receives the first client request and redirects it to the suitable least-loaded receiver. And, when a member exceeds its *critical-threshold* the load manager alerts it, then the member transfers the next request back to the load manager to be redirected again to the suitable least-loaded receiver. Hence, LL is considered a centralized strategy from the transfer perspective.

Notice that the load manager determines which members can participate in a load transfer at two times:

1. When the first request of a client arrives to the load manager.
2. After a member exceeds its *critical-threshold*.

The first time indicates that LL is event-triggered, but the second time depends on the implementation and this aspect is not ruled by the specification. For example, following the execution of a pull-style periodic information policy, the transfer policy would be periodic. On the other hand, if the transfer policy asynchronously notifies that a member has exceeded its *critical-threshold*, then the transfer policy would be event-triggered.

2.3 Selection policy

LL uses a non-preemptive selection policy. After a member exceeds its *critical-threshold*, it will redirect the next request to the load manager. It means that a task is not interrupted after it initiates because the decision of rejecting the next request is taken before it arrives to the overloaded member. It leads to a simpler service design and implementation because it does not require a mechanism to stop the execution of a method and transfer the intermediate state of the object to another node.

2.4 Location Policy

LL simply selects the suitable least loaded group member as receiver of a task. A suitable member has an effective load value that does not exceed the *reject-threshold*. If several group members are suitable and have the same lowest effective load for several consecutive requests, requests are handed to these members in least-recently used order.

3 CORBA Enhanced Least-Load Strategy

CORBA least-loaded strategy (LL) is an effective load-balancing algorithm that typically offers better average performance than static strategies (e.g. random and round-robin), and presents little overhead. However, LL does not account for: i) heterogeneous multicomputers systems, ii) inactive nodes, and iii) a robust specification of the pull period of the information policy. The proposed eLL overcomes these pitfalls.

3.1 Heterogeneous Multicomputer Systems

LL assumes that group members are allocated at nodes with the same processing capacity (as usual). It considers the load of the nodes but not their performance differences. Considering all the nodes as homogeneous, we lose the chance to allocate more work at cluster nodes with better performance (i.e. faster nodes).

Load-balancing systems with heterogeneous nodes are not uncommon. In many cases, a load-balanced cluster initially has identical computers, but later the demand rises and additional computers must be incorporated. These computers typically have better performance than the original ones. Therefore, it is important that a load balancing strategy effectively addresses this situation.

For this purpose, we include the Performance Index (PI) that represents a measure of the performance of a node. This index must be greater than 0.0 and is inversely proportional to the node performance, so lower values correspond to faster nodes. PI may be estimated executing performance tests on nodes (e.g. determining the processing time of specific tasks). Then, the effective load is calculated as follows:

$$el_k = \left\lfloor \frac{sf}{tol} \cdot (A + B) \right\rfloor \tag{2}$$

$$A = d \cdot \left(el_{k-1} \cdot \frac{tol}{sf} + r_{k-1} \cdot pbl \right)$$

$$B = (1 - d) \cdot nrl$$

$$sf = \frac{PI_n}{PI_{min}}$$

$$PI_{min} \leq PI_n \Rightarrow sf \geq 1.0$$

where PI_n is the performance index of the *n-th* cluster node, PI_{min} is the lowest performance index and sf is the scaling factor (the other terms are described in Section 2.1). This means that if we have nodes A and B with the same load (e.g. similar CPU queue length), and A is 20% faster than B, then the strategy would consider A approximately 20% less loaded than B (this approximation is the result of the integer division).

3.2 Detection of Inactive Nodes

LL is a sender-initiated algorithm whose transfer policy is controlled by the *reject-threshold* and *critical-threshold*. When the member load goes beyond the former (the member is attending a considerable amount of work), the member becomes a non-suitable receiver. And, when the latter is exceeded (the member is overloaded), the member sends back the next request to the load manager.

This transfer policy has proved to be effective dealing with considerably loaded and overloaded members. However, it lacks of a mechanism to detect inactive and/or slightly loaded nodes, so it does not take full advantage of the processing capacity of the nodes. For example, when the load-balancing cluster includes heterogeneous nodes, the processing time of tasks may widely vary from node to node; thus, some nodes may be non-active while other nodes are considerably loaded. Similar results are obtained when cluster nodes run additional non-balanced tasks, or the processing time of tasks (e.g. a method) strongly depends on its arguments.

To prevent this situation we transform LL to a symmetrical-initiated algorithm [17]. Therefore, when a node is non-active the load manager instructs other nodes to transfer load to it. Two additional thresholds have been included to control this symmetrical-initiated transfer policy. These are:

1. *idle-threshold*: if the effective load value of a group member is lower than or equal to this threshold then the strategy will consider this member as idle.

2. *busy-threshold*: if the effective load value of a group member is greater than or equal to this threshold then the strategy will consider this member as busy. A busy member may be selected to transfer its load to non-active members.

If *idle-threshold* and *busy-threshold* are both non-zero, *busy-threshold* must not be smaller than *idle-threshold*.

In addition to the original sender-initiated part of the transfer policy, a possible implementation of the new receiver-initiated part might be as follows:

1. The algorithm periodically checks the member loads and counts the non-active nodes.

2. With this information a list of busy nodes is built, and the algorithm sorts it in descendent order according to the effective load of the members.

3. Finally, the load manager goes through the list and instructs some busy nodes to transfer upcoming requests. The number of nodes notified is the lower number of idle and busy nodes.

3.3 Mixed-Style Load Monitoring

The proposed Load Balancing and Monitoring Service specification [8] allows two styles of load monitoring: pull and push (see Section 2.1). Pull style is typically periodic while push style is threshold-driven. As previously stated, the main shortcoming of the periodic pull style is that the user must set an appropriate period, whereas using push style the load information could risk being non-updated and more elaborated load monitors are required.

This paper proposes to combine both monitoring styles. The periodic pull style is configured with a relative long period, and when a member (specifically, a load monitor) detects an important change on its load index(es), it notifies its load information to the load manager.

Load monitors detect load changes using thresholds. These thresholds are equivalent to those specified by eLL: *idle*, *busy*, *reject* and *critical* (see Sections 2.2 and 3.2). Notice that eLL and load monitor thresholds are relative to effective and raw load values, respectively. Therefore, load monitor thresholds are calculated as follows (derived from Equation 2):

$$TH_{lm} = \frac{tol}{sf} \cdot TH_{st} \tag{3}$$

where TH_{lm} is the load monitor threshold, TH_{st} is a eLL threshold, tol is the tolerance, and sf is the scaling factor.

The additional load monitor thresholds do not significantly increase the complexity of the system. These thresholds may be set on each load monitor either manually by an administrator (e.g. using load monitor configuration files) or automatically by the load manager (e.g. when the group registration occurs). In the automatic case, the `LoadMonitor` interface must be extended to include few additional administrative operations.

This mixed approach actually keeps load information updated and lightly increases the network traffic.

4 Case Study

Homogeneous multicomputer systems are typically represented by the Multiple Queue/Multiple Server (MQMS) model. However, this model generally becomes analytically intractable under load balanced heterogeneous systems with arbitrary service time distribution [2]. Because of this intractability, comparative studies of load balancing algorithms are made using simulation tools [2, 17, 5, 1]. Consequently, a discrete-event simulation tool, coded in Java using object-oriented programming techniques, was developed to compare the performance of the least-loaded (LL) and enhanced least-loaded (eLL) algorithms. The tool emulates the proposed CORBA Load Balancing and Monitor Service [8]. The round robin (RR) algorithm is also included in the study as a reference.

Our study includes several node configurations with different heterogeneity levels. This paper presents the two extreme configurations considered: a homogeneous case and a highly heterogeneous case (Table 1). These cases are representative of the remaining configurations and their omission does not affect the conclusions of this work.

Table 1: Case Configurations

Cases\Nodes	N1	N2	N3	N4	N5
1	C	C	C	C	C
2	C	C	B	B	A

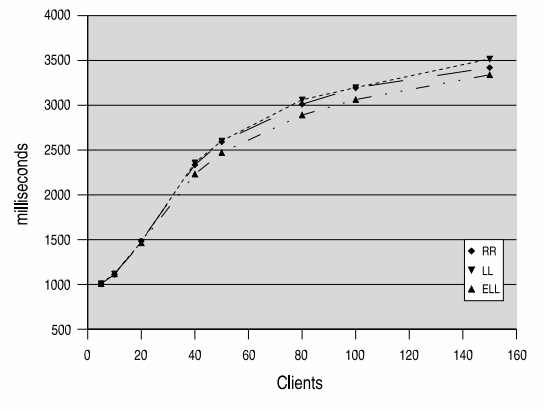
In the Table 1, A, B and C denotes the different types of nodes, where A nodes are 30% faster than B nodes, and B nodes are 40% faster than C nodes; thus, the execution time of a request mainly depends on the processing capacity of the node that processes the request. Notice that all cases use five nodes and each node holds only a group member.

The methodology tests each case using 5, 10, 20, 40, 50, 80, 100 and 150 clients. All clients start at the same time and each one sequentially sends synchronous requests. An exponential distribution ($\mu = 2.5\,sec.$) determines the interval of time between consecutive requests. Furthermore, *Poisson* distributions are used to generate the number of requests per client ($\mu = 7$) and the request-processing time of the nodes ($\mu_A = 420$, $\mu_B = 600$, $\mu_C = 1000$ in milliseconds). The remaining simulation parameters are: *critical-threshold* $= 16$, *reject-threshold* $= 12$, *busy-threshold* $= 2$, *idle-threshold* $= 1$, $tol = 1$, $d = \frac{1}{4}$ and $pbl = 0$. Communication overhead is considered negligible except the overhead caused by load transfers ($50\,msec.$).

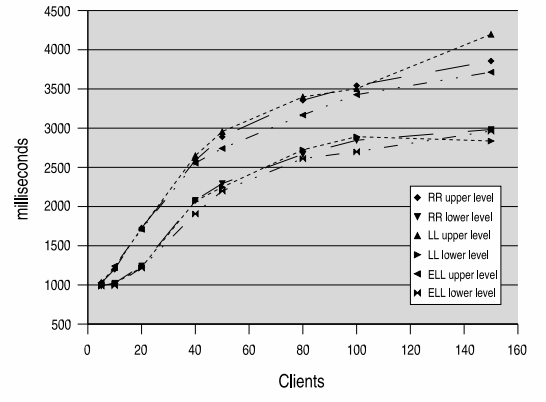
The performance metrics used in the tests are the Average of the Mean Request Response Time (AMRRT) and the 99% AMRRT Confidence Interval (ACI). These metrics are calculated from 30 runs of each case.

5 Results and Discussion

Figures 1 and 2 show the simulation results for each case. The results indicate that eLL outperforms the other algorithms, particularly when the load is high and the nodes are heterogeneous; eLL has the best AMRRT when there are more than 40 clients in Case 1, and 20 clients in Case 2. Specifically, the AMRRT of eLL is as much as 5% and 15% lower than the AMRRT obtained using LL in the cases 1 and 2, respectively.



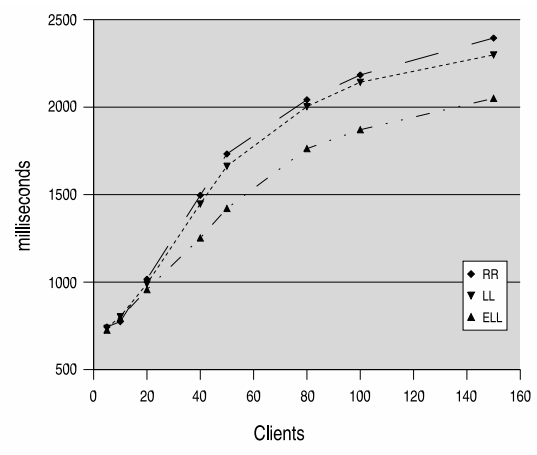
(a) Average of the Mean Request Response Time (AMRRT)



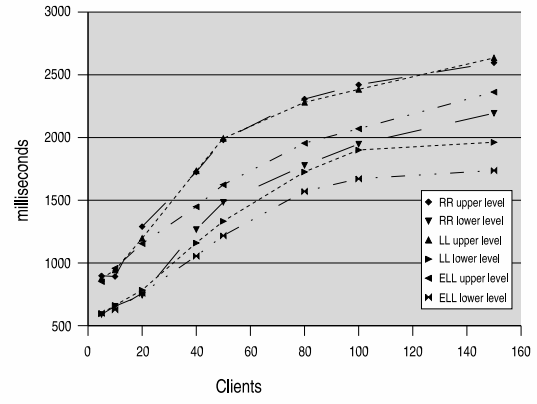
(b) AMRRT Confidence Interval (ACI)

Figure 1: Simulation Results (Case 1).

On the other hand, all strategies have similar ACI in Case 1. However, the predictability of eLL lightly improves in Case 2 in which the ACI of eLL reaches up to 38% lower than the ACI of LL. Notice that the algorithms have similar behavior when the number of clients is low, and the ACI rises as the load increases. It is necessary to emphasize that Case 1 is the most favorable case for RR, due to the computer homogeneity.



(a) Average of the Mean Request Response Time (AMRRT)



(b) AMRRT Confidence Interval (ACI)

Figure 2: Simulation results (Case 2).

6 Conclusions and Future Work

This paper presented a load balancing strategy, called enhanced least-loaded (eLL), that has potentially better average performance than the CORBA least-loaded (LL) strategy, a built-in dynamic algo-

rithm for the proposed standard CORBA Load Balancing and Monitoring Service. The eLL takes into account: i) heterogeneous multicomputer systems, ii) detection of inactive nodes and iii) a mixed-style load notification. Considering the Average of the Mean Request Response Time (AMRRT) and the AMRRT Confidence Interval (ACI), preliminary simulation results have showed that eLL has an AMRRT (ACI) up to 15% (38%) lower than the AMRRT (ACI) obtained using LL. In addition, this work allowed us to verify how eLL takes advantage of differences in node processing capacity, specially when load is high.

Future work will include:

- a comparative study of eLL and LL using a more sophisticated simulation-centered methodology based on statistics of real systems.
- a comparison of eLL and other load balancing algorithms suitable to be incorporated in the upcoming CORBA Load Balancing and Monitoring Service.
- a stability study of eLL.

Acknowledgements

The authors gratefully acknowledge the financial support provided by the Fondo Nacional de Ciencia, Tecnología e Innvovación (FONACIT-Venezuela) through the Project G-97000824.

References

[1] S. Albers and B. Schröder. An Experimental Study of Online Scheduling Algorithms. *Lecture Notes in Computer Science*, 1982:11–25, 2001.

[2] S. A. Banawan and N. M. Zeidat. A Comparative Study of Load Sharing in Heterogeneous Multicomputer Systems. In *25th Annual Simulation Symp.*, pages 22–31, USA, Apr. 1992. IEEE.

[3] Distributed Object Computing (DOC). TAO Overview. `http://www.cs.wustl.edu/~schmidt/TAO-intro.html`.

[4] R. L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[5] C. C. Hui and S. T. Chanson. Improved Strategies for Dynamic Load Balancing. *IEEE Concurrency*, 7(4):58–67, Sept. 1999.

[6] IONA Technologies PLC. *Load Balancing and Monitoring. Initial Joint Submission (orbos/01-08-05)*, Aug. 2001.

[7] IONA Technologies PLC. *IONA Orbix E2A CORBA Technology. White Paper*, Apr. 2002.

[8] IONA Technologies, Tri-Pacific Software Inc. and VERTEL Corporation. *Load Balancing and Monitoring. Revised Joint Submission (mars/02-04-05)*, Oct. 2002.

[9] M. Lindermeier. Load Management for Distributed Object-Oriented Enviroments. In *2nd International Symposium on Distributed Objects and Applications (DOA 2000)*, pages 59–, Antwerp, Belguim, Sept. 2000. OMG.

[10] Object Management Group, Inc. *Load Balancing and Monitoring for CORBA-based Applications. Request for Proposal (orbos/2001-04-27)*, Apr. 2001.

[11] Object Management Group, Inc. *Common Object Request Broker Architecture: Core Specification (Version 3.0)*, Nov. 2002.

[12] O. Othman, C. O'Ryan, and D. C. Schmidt. An Efficient Adaptive Load Balancing Service for CORBA. *IEEE Distributed Systems Online*, 2, Mar. 2001.

[13] O. Othman, C. O'Ryan, and D. C. Schmidt. The Design of an Adaptive CORBA Load Balancing Service. *IEEE Distributed Systems Online*, 2, Apr. 2001.

[14] L. P. Peixoto dos Santos. Load Distribution: a Survey. Technical Report UM/DI/TR/96/03, Departamento de Informática. Escola de Engenharia. Universidade do Minho, Oct. 1996.

[15] PrimsTech. OpenFusion Load Balancing Service v2.5 - White Paper. `http://www.prismtechnologies.com/English/Products/index.html`, Sept. 2002.

[16] A. Puder and K. Roemer. *MICO: An Open Source CORBA Implementation.* Morgan Kaufmann, 3rd edition, Mar. 2002. ISBN: 1558606661.

[17] N. G. Shivaratri, P. Krueger, and M. Singhal. Load Distributing for Locally Distributed Systems. *IEEE Computer*, 25(12):33–44, Dec. 1992.