A New Implementation for Parallel Processing Based on CORBA Standard

Ricardo R. dos Santos, Danielle P. de Ruchkys, Milton E. R. Romero, Luciano Gonda, Amaury A. C. Junior
Dom Bosco Catholic University
Computing Engineering Department
Av. Tamandaré, 6000 – CEP – 79117-900 – Campo Grande (MS) – Brazil
Phone/Fax: (55) 67 312-3800
{ricrs,danielle,miltonr,gonda,amaury}@ec.ucdb.br – Speaker: Ricardo Ribeiro dos Santos

Abstract: In this work we present a new implementation based on Common Object Request Broker Architecture (CORBA) to develop parallel applications. This approach is based on asynchronous messaging, using Asynchronous Messaging Invocation (AMI) models, along with an extension of the Implementation Repository to manage and distribute remote processes. Two interesting characteristics are suported here: first, to allow the building of CORBA applications with parallel resources and suitable performance; and second, to provide interoperability among different Object Request Brokers (ORBs) and applications so that structure of CORBA architecture remains unmodified.

Keywords: CORBA, parallel applications, performance, interoperability, message passing.

1. Introduction

The Common Object Request Broker Architecture (CORBA) [1,2,3] is one of the most known and used architecture to help in object oriented distributed applications development. The CORBA architecture specification defines how objects can communicate and interact in computer networks. However, communication and interaction among processes are not the only important features in network applications, in particular, in parallel applications on clusters, computing performance is also an issue.

PARallel DIStributed (PARDIS) [5,6], Parallel CORBA (PaCO) [8,9] and Parallel Component Bus (PCB) [7] perform parallel computing based on CORBA. PARDIS implements application-level interaction of heterogeneous, parallel components in a distributed environment [5] by extending the CORBA object model and IDL[6] introducing Single Program Multiple Data (SPMD) objects. PCB is based on the Java-CORBA technologies, suitable for SPMD and MIMD models [7]. PCB considers the processes scheduling but the parallelism is not transparent to the user application. PaCO is an evolution of the COBRA system [8] with the goal of hide the parallelism from the user [9]. In PaCO each CORBA object encapsulates a SPMD process of the parallel code maintaining interoperability as it does not modify the CORBA architecture, but as well as in PARDIS, it does not support a processes scheduling mechanism [9]. Patterns like Asynchronous Messaging Invocation (AMI) [4] has been developed to improve the CORBA applications performance. However, resources that enable remote application management are still a need to be addressed in order to let the CORBA environment to be used as a suitable tool for parallel applications development.

First intuition behind this work regards the ability to address real problems. Our current parallel computer environment for coding and energy optimization applications is based on coordination languages on clusters. Thereby, this work broadens the environment as it allows the building of CORBA applications with parallel resources maintaining interoperability among different ORBs and applications in such a way that structure of the CORBA architecture remains unmodified. Second intuition is how to schedule and manage remote processes with a suitable performance. Thus, this work presents a new implementation that uses: 1) AMI model [4] as the asynchronous communication feature, and 2) the Implementation Repository of the ACE ORB (TAO) implementation [11] for applications management and scheduling. Note that these issues are also partially addressed by other CORBA tools. However, interoperability among different ORBs and applications in such a way that structure of the CORBA architecture remains unmodified and scheduling and managing of remote processes with a suitable performance are simultaneous characteristics presented in our implementation.

In the next section, we describe the CORBA architecture. Section 3 presents a brief description of three approaches of parallel computing on CORBA. Section 4 addresses the characteristics of our implementation. Section 5 discusses the new implementation in comparison with previous approaches and show performance comparison between coordinationation languages and the proposed implementation. Finally, section 6 sumarizes some conclusions and future work.

2. CORBA Architecture

CORBA is a standard middleware for communication that provides a set of services allowing the distribution of objects through messages, among a set of computing resources connected to a common network.

The Object Request Broker (ORB) that mediates communication between client and server objects independently of the underlying network is known as the core of the CORBA architecture. All of the requests to (and replies from) distributed objects are passed through the ORB [1].

CORBA ORB includes the General Inter-ORB Protocol (GIOP). GIOP specifies a standard transfer syntax (low level data representation) and a set of message formats for communications between ORBs. To facilitate the internetworking of commercial ORBs developed by different vendors, the CORBA 2.0 standard defined the Internet Inter-ORB Protocol (IIOP) [3]. IIOP allows that objects operating over heterogeneous IIOP-compliant ORBs interact with each other through a simple and generic interface, regardless of the internal structure of the ORBs or of any other vendor-specific mechanisms.

In Figure 1 the CORBA architecture is presented. At the client side (shown as Master in Figure 1), the dynamic interface is the DII (Dynamic Invocation Interface) and at the server side (shown as Slave in Figure 1), the dynamic interface is the DSI (Dynamic Skeleton Interface). Dynamic creation and invocations to objects are provided by the DII, while a way to deliver requests from an ORB to an object implementation that does not have compile-time knowledge about the type of the object is provided by the DSI. Rather than using the dynamic interfaces, that are complex to utilize, a static interface through the Interface Definition Language (IDL) is provided by CORBA.

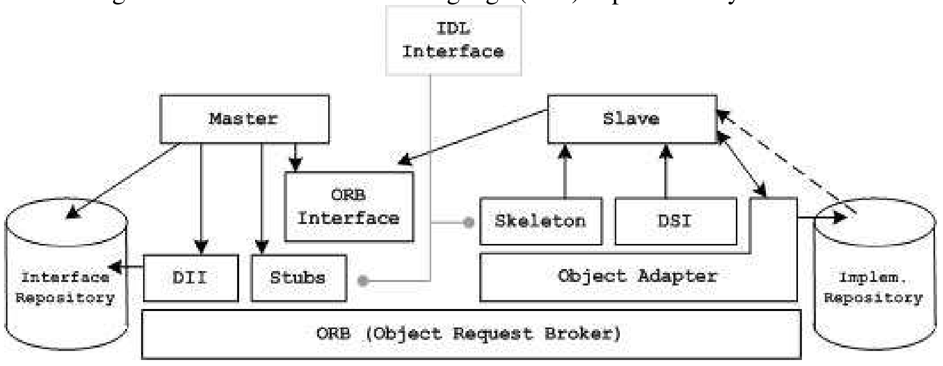

Figure 1. CORBA Architecture

For a given object, an IDL file contains a list of operations that can be remotely invoked. The definition of available types in the parameter specification for each operation is an IDL concern. Generation of a stub at the client side and a skeleton at the server side is an IDL compiler job. Stubs and skeletons aim at connecting a client of a particular object to its implementation through the ORB. Stubs and skeletons are often tied to a specific CORBA implementation; thereby, they are not always portable.

The client calls its local stub in each invocation. The client side stub creates a message with method identification and parameters and sends it to the remote server that replies with a message that is forwarded to the client by the client's stub.

The skeletons offer an interface to each implemented method on the server. The skeleton that calls the local server receives the message with method identification and parameters. The obtained results are marshalled by the skeleton and are sent to the client. Note that the client and the server objects do not know the remote placement of each other.

Several problems when dealing with distributed objects, such as transparent activation of objects with different policies, are addressed by the Object Adapter that provides a suitable interface to work on, as it is shown in Figure 1.

Data services described in the interfaces are stored in the metadata base named Interface Repository (IR). This repository is, basically, used by the DII in its invocations. On the other hand, the Implementation Repository (named Implem. Repository in Figure 1) stores information about the supported classes in the server. The new server object instances needed are created by the IR to receive client requests.

In Figure 1, the ORB Interface is a set of functions that can be used by both the client and the server objects to perform suitable operations, such as, convert strings to object references and vice-versa. We kindly refer the interested reader to [1] for the details.

3. Related Works

In order to perform parallel computing based on CORBA three approaches were presented in literature: PARDIS [5,6], PaCO [8,9] and PCB [7]. PARDIS was described among the first CORBA-based systems to implement application-level interaction of heterogeneous, parallel components in a distributed environment [5]. PARDIS extends the CORBA object model and IDL by introducing Single Program Multiple Data (SPMD) objects representing data-parallel computations. PARDIS distinguishes two kind of objects: SPMD objects and single objects. Each SPMD object is associated with one or more computing threads that are capable of performing services. Single objects are associated with only one computing thread. The CORBA IDL is extended to provide data distribution among different threads associated with a SPMD object. SPMD objects can use distributed argument types in operation definitions. In order to do this, PARDIS provides a generalisation of the CORBA sequence: the distributed sequence. The invocations in PARDIS can also be non-blocking, using the future, that is a way to represent results that are not yet available. PARDIS maintains interoperability because it does not modify the CORBA architecture, but it does not support a processes scheduling mechanism [5].

PCB is described as an architecture for distributed computing, based on the Java-CORBA technologies, suitable for SPMD and MIMD models [7]. The fact of the slaves and masters objects need to be manually initialized one-by-one on the cluster of computers is pointed as an important issue in distributed system to be used in parallel applications. To address this problem and to provide some other facilities PCB adds a layer (PCB layer) between the CORBA layer and the application layer. PCB has a component bus that is an extension of the CORBA bus. Each SPMD computation is associated with a component bus, and each MIMD computation consists of multiple sequences of control that require multiple component buses. It is important to note that PCB is the only system in this section that considers the processes scheduling. Another feature is that the parallelism is not transparent to the user application because the non-blocking communication mechanism is not defined by the CORBA standard [7].

PaCO is also intended for parallel programming and it is an evolution of the COBRA system [8] with the goal of hide the parallelism from the user [9]. The concept of parallel object in PaCO is a collection of identical standard CORBA objects. Each CORBA object encapsulates a SPMD process of the parallel code. When a client invokes a remote operation in a parallel CORBA object the associated method is concurrently executed by all objects belonging to the collection. Such parallel execution is performed under the control of the stub associated with the parallel CORBA object. Because the stub behaves differently from the one associated with a standard CORBA object, the way an IDL compiler generates stubs was modified. This IDL modified by PaCO is called Extended IDL (EIDL). It is important to note that in PaCO (as well as in PARDIS) the data distribution is specified in the IDL. PaCO also maintains interoperability because it does not modify the CORBA architecture, but as well as in PARDIS, it does not support a processes scheduling mechanism [9].

4. A New Implementation For Parallel Processing Based on CORBA Standard

A new approach that provides parallel computing resources to CORBA applications without changes in the Interface Definition Language (IDL) of the CORBA standard [10] is here presented. To provide interoperability among ORBs, the CORBA standard architecture is kept unmodified.

The solution here developed is based on the management of the remote CORBA applications and on the asynchronous communication through AMI models. The management of the applications is performed by the use of the Implementation Repository of The ACE ORB (TAO) tool [11]. The asynchronous communication feature and the better performance in comparison with other communication mechanisms in CORBA [12,13] motivate the utilization of AMI models in the proposed implementation. Support to AMI models through callback invocations provided by TAO motivated the utilization of this tool in the proposed implementation.

TAO originally can not support remote processes management, therefore, TAO's IR is modified to cope with two main needs: (1) initialization and distributed IRs management; (2) scheduling processes in a cluster environment.

In order to cope with the first requirement, all IRs develop the same functions and to cope with the second requirement, the implementation of a scheduler mechanism and a round-robin scheduling policy are included.

Multiple Program Multiple Data (MPMD) in which a parallel application is composed by the master and the slave processes, is the adopted paradigm in this implementation. Master processes send/receive data to/from slave processes. Slave processes implement a parallel algorithm and are responsible for computing the received data. Note that the master and the slave processes perform the same tasks as the client and the server components, respectively. Figure 1 presents the CORBA architecture with the master and the slave processes. The new communication interface (dotted line) from the Implementation Repository to the slave process can be observed. This new communication interface is used to schedule and manage slave processes executing in remote hosts. Note also that, IR is extended to support these features. As they do not modify IDL language or the ORB of the CORBA architecture, then, TAO maintains interoperability with default CORBA applications.

Figure 2 presents a step by step description to schedule slave processes. First, the master process calls the local IR to find the list of machines that will participate of the scheduling. Then, the local IR searches its hosts list (using a round-robin scheduling policy) and returns host names that should be used in the scheduling. In next step, master process invokes the *spawn_app* method to start slave processes. This invocation is sent to local IR that forwards it to remote hosts.

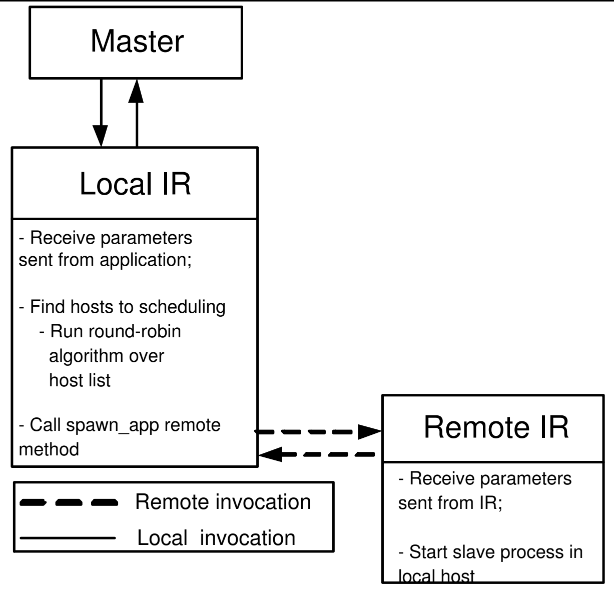
Master
Local IR
- Receive parameters sent from application;
- Find hosts to scheduling
- Run round-robin algorithm over host list
- Call spawn_app remote method
Remote IR
- Receive parameters sent from IR;
- Start slave process in local host
Remote invocation
Local invocation

Figure 2. Steps to schedule processes

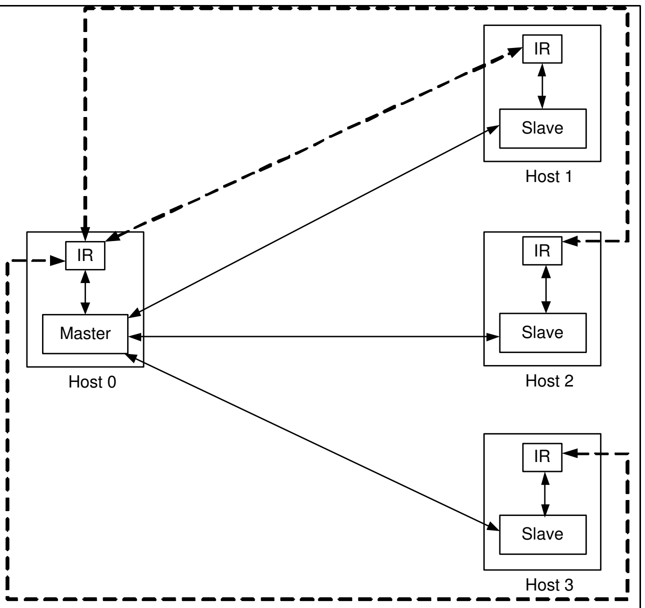

Figure 3. Communication among IRs and processes

Among remote IRs in the cluster environment there is a direct interaction. This interaction is mainly used to initialization of remote IRs and schedule slave processes. In Figure 3, the dotted lines indicate this interaction, while solid lines indicate method invocations among processes of the parallel application. Note that, the Implementation Repository of the TAO was, originally, organized in a centralized way. By adding management and scheduling features to the processes, a distributed organization with an IR in each host of the cluster was adopted. Therefore, a faulty IR does not affect processes running in other hosts as a consequence of the distributed organization.

5. Implementation Discussion

First intuition behind this work regards the ability to address real problems. Our current parallel computer environment for coding and energy optimization applications is based on coordination languages on clusters as PVM and MPI. Thereby, this work broadens the environment as it allows the building of CORBA applications with parallel resources maintaining interoperability among different ORBs and applications in such a way that structure of the CORBA architecture remains unmodified. Second intuition is how to schedule and manage remote processes with a suitable performance.

Note that these issues are also partially addressed by other CORBA tools. Regarding interoperability, as the non-blocking communication mechanism used by applications is not defined by the CORBA standard in PCB, the data parallelism is not transparent to the user application. On the other hand, PARDIS and PACO approaches maintain the interoperability feature because they do not modify the structure of CORBA architecture and allow transparent data parallelism that is built from IDL statements. In these solutions stubs and skeletons are modified in order to build the parallel processing capabilities.

Our approach only extends the CORBA architecture in a different way as PARDIS and PaCO have done; in particular, we extend the services on the Implementation Repository (IR). It is worth noting that, to the best of our knowledge, our implementation is the only one based on non-blocking communication through AMI model providing interoperability. Note also, that in our approach the parallelism must be explicitly defined in the client code. As changes in IDL file are not necessary, the interface keeps the portable feature of the CORBA architecture.

Regarding processes scheduling, PARDIS and PaCO are solutions to specific hardware architectures and do not support a processes scheduling mechanism, on the other hand, PCB implements processes scheduling, as well as our proposal.

Regarding performance, as our current environment is based on LAM-MPI and PVM coordination languages, the CORBA implementation has shown comparable (comunication latency) performance with respect to the current environment as shown in Figure 4 for illustration purposes. Thereby, it can be considered as a likely way suitable for a parallel computing environment, at least, taking into account our preliminary results. Migration from current coordination languages to CORBA support for parallel computing for real applications is not an easy and simple decision to take from the practical point of view.

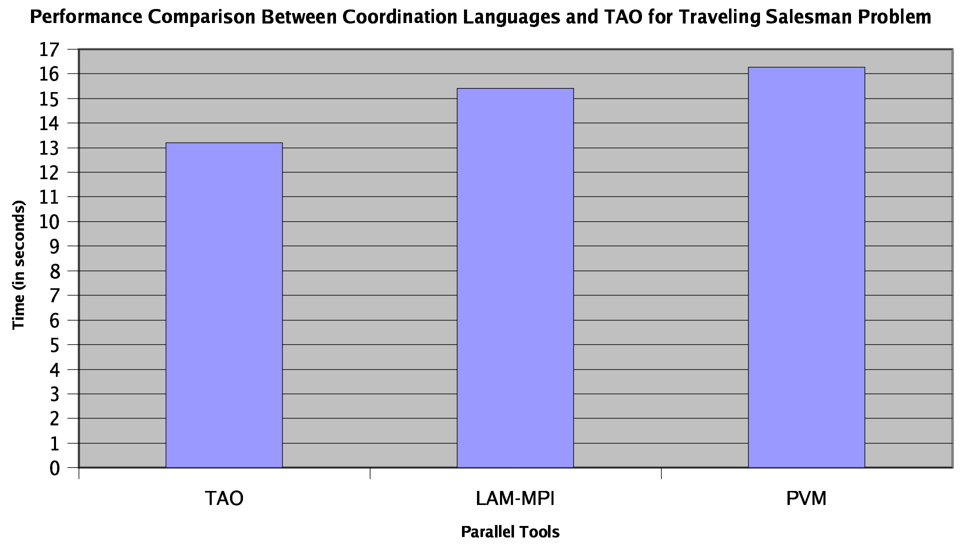

Figure 4. Performance comparison between coordination languages and TAO

6. Conclusions

We have showed how to utilize CORBA as a possible architecture to create a framework to develop parallel applications. Challenges to CORBA parallel processing tools developers are twofold: 1) to allow the building of CORBA applications with parallel resources and suitable performance; and 2) to provide interoperability among different ORBs and applications. We have presented how to build those features in such a way that structure of CORBA architecture remains unmodified. A structure to manage distributed processes in such a way that modifications of IDL file, stubs, or skeletons are needless has been here presented. Basically, we have approached these challenges by extending the services of the Implementation Repository and by using the resources of the asynchronous communication CORBA standard to develop invocations between client and server programs, thereby, the focus is on transparent code distribution and management.

A comparison among features of available tools that provide parallel computing based on the CORBA architecture has been also here presented. We have also presented preliminary performance communication results of client/server invocation that likely let the CORBA environment be a suitable tool for parallel processing. Preliminary results have shown that parallel computing specific tools are comparable with the CORBA tool here proposed regarding the parallel communication application performance. However, in future work, we intend to address, more carefully, the performance comparisons among standard parallel tools and this proposal, in particular, comparing the processing of our current applications in coding and energy optimization.

References

[1] Mowbray, T. J., Malveau, R. C.: CORBA Design Patterns. John Wiley Sons. (1997)

[2] Mowbray, T. J., Ruh, W. A.: Inside CORBA: Distributed Object Standards and Applications. Addison-Wesley. (1997)

[3] Vinoski, S., Henning, M.: Advanced CORBA Programming with C++. Addison-Wesley. (1999)

[4] Schmidt, D. C., Vinoski, S.: An Introduction to CORBA Messaging. C++ Report Magazine. **15** (1998)

[5] Keahey, K.: PARDIS: A Parallel Approach to CORBA. Proceedings of the 6th IEEE Internation Symposium on High Performance Distributed Computing. August (1997)

[6] Keahey, K., Gannon, D.: PARDIS: CORBA-Based Architecture for Application-Level Parallel Distributed Computation. Proceedings of SC97 (Supercomputing '97). November (1997)

[7] Xu, C. W., He, B.: PCB - A Distributed Computing System in CORBA. Journal of Parallel and Distributed Computing. **60** (2000) 1293--1310

[8] Beaugendre, P., Priol, T., René, C.: COBRA: ACORBA-Compliant Programming Environment for High Performance Computing. Technical Report, Institut de Recherche en Informatique et Systèmes Aléatoires. **1141** (1998)

[9] Denis, A., Pérez, C., Priol, T. Achieving Portable and efficient Parallel CORBA Objects. Concurrency and Computation: Practice and Experience. **15** (2003) 891--909

[10] Santos, R. R.: Parallel Applications Scheduling: AMIGO-CORBA Interface. Masther's Dissertation. Institute of Mathmatical and Computer Sciences, University of Sao Paulo, Brazil. (2001)

[11] Schmidt, D. C., Vinoski, S.: Modeling Distributed Object Applications. C++ Report Magazine. **2** (1995)

[12] Zuffo, F., Santos, R. R.: Analysis and Performance Evaluation of CORBA Communication Mechanisms. Proceedings of XI SIICUSP, São Paulo, Brazil. (2003)

[13] Arulanthu, A. B. and Schmidt, D. C. and O'Ryan, C. and Kircher, M. and Parsons, J.: The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging. Proceedings of the IFIP/ACM Middleware Conference. (2000)

[14] OMG: Data Parallel CORBA. RFP - Document orbos/01-06-08. (2001)