ELSEVIER

Future Generation Computer Systems 18 (2002) 841–847

FGCS
FUTURE
GENERATION
COMPUTER
SYSTEMS

www.elsevier.com/locate/future

Tool environments in CORBA-based medical high-performance computing

T. Ludwig [a,*], M. Lindermeier [b], A. Stamatakis [b], G. Rackl [b]

[a] Institut für Informatik, Ruprecht-Karls-Universität Heidelberg, Im Neuenheimer Feld 348, 69120 Heidelberg, Germany
[b] Technische Universität München, Institut für Informatik, 80333 Munich, Germany

Abstract

High-performance computing in medical science has led to important progress in the field of computer tomography. A fast calculation of various types of images is a precondition for statistical comparison of big sets of input data. With our current research we adapted parallel programs from PVM to CORBA. CORBA makes the integration into clinical environments much easier. In order to improve the efficiency and maintainability we added load balancing and graphical on-line tools to our CORBA-based application program. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Computer tomography; Parallel computing; Load balancing

1. Introduction

Imaging in medical science is an important issue that shows an increasing connection with high-performance computing. Relevant picture series from imaging hardware like magnetic resonance tomographs or positron emission tomographs are usually computed on powerful servers and stored in specialized picture archiving systems.

Recently, workstation clusters became more and more popular as they provide a good price–performance ratio. Furthermore, many operations that are performed on these picture series exhibit a maximum parallelism. In many cases no interprocess communication is required and the parallelization is handled at the granularity level of the individual pictures.

As soon as the parallel imaging servers are used in production mode we are faced with two more problems. One is the load of the individual nodes of the cluster. It should be balanced in order to guarantee an optimal use of the computational power of the cluster. Second, the imaging software has to interact with other software components in a medical environment and thus has to meet certain standards of reliability and interoperability.

The paper will present an approach where we base our parallelization of the imaging software on a distributed object-oriented middleware system (in our case CORBA) to take advantage of component integration. A load balancing mechanism is integrated into a specific CORBA ORB to provide optimal performance to the application programs.

2. The load management system

Load management systems can be classified according to their implementation. They may be integrated into the application, the runtime system, or a separate service. The first case is called application level,

* Corresponding author.
E-mail address: thomas.ludwig@informatik.uni-heidelberg.de (T. Ludwig).

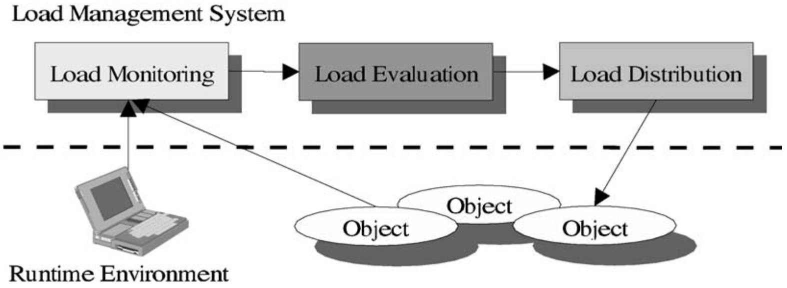

Fig. 1. The components of a load management system.

the second one system level, and the third one service level load management. We decided to make a system level implementation because it provides maximum flexibility and transparency to the user.

In general, load management systems can be split into three components: The load monitoring, the load distribution, and the load evaluation component. They fulfill different tasks and work at different abstraction levels. This eases the design and the implementation of the overall system. Fig. 1 shows the components of a load management system and a runtime environment containing some application objects.

The load monitoring component provides both, information on available computing resources and their utilization, and information on application objects and their resource usage. This information has to be provided dynamically, i.e. at runtime, in order to obtain knowledge about the runtime environment and its objects. The computing resources in distributed environments may be shared by middleware-based applications and legacy applications.

Load distribution provides the functionality for distributing workload. Load distribution mechanisms for system level load management are initial placement, migration, and replication.

- *Initial placement* stands for the creation of an object on a host that has enough computing resources in order to efficiently execute an object. Initial placement may be applied to all kinds of objects because it is done at creation time.
- *Migration* means the movement of an existing object to another host that promises a more efficient execution. It may be applied to all kinds of objects, too. However, migration is applied to existing objects, so the object state has to be considered. The object's communication has to be stopped and its state has to be transferred to the new object. Finally, all communication has to be redirected to the new object.
- *Replication* is similar to migration but the original object is not removed, so some identical objects called replicas are created. Further requests to the object are divided up among its replicas in order to distribute workload (requests) among the replicas. Replication is restricted to replication safe objects. This means that an object can be replicated without applying a consistency protocol to the replicas. A precise definition of the term replication safe can be found in [7].

Finally, the load evaluation component makes decisions about load distribution based on the information provided by load monitoring. The decisions can be reached by a variety of strategies. The aim of the diverse strategies is to improve the overall performance of the distributed application by compensating load imbalance. There are two main reasons for load imbalance in distributed systems. First, background load can substantially decrease the performance of a distributed application. Second, request overload that is caused by too many simultaneously requesting clients increases the request processing time and thus, decreases the performance of the overall application. Both sources of load imbalance have to be considered by a load management system.

Distributed object-oriented environments like CORBA [10] or DCOM [2] are based on some kind of object model. In general, the object models imply some transparency requirements [8]. Location transparency demands that the location of an object is unknown to its user. The middleware transparently connects client and server. Access transparency postulates that all objects in a distributed system are accessed in the same way. The middleware is responsible for providing uniform access to all objects, independent of their implementation or runtime environment. These transparency requirements have to be fulfilled by load management systems, too. Therefore, load distribution has to be transparent to the user. Our load management system provides full migration and replication transparency which means that migration and replication are completely transparent to the user.

The load management concepts described so far are universal and may be applied to diverse distributed object-oriented environments. The implementation of these concepts strongly depends on the underlying middleware architecture. We decided to make an implementation for CORBA because it is the most popular middleware architecture.

In CORBA, objects are connected to the middleware by the POA (Portable Object Adapter). The object adapter provides the functionality for creating and destroying objects, and for assigning requests to them. The POA is configured by the developer via the so-called policies. The ORB (Object Request Broker) provides the functionality for creating object adapters and for request handling. A request to an object arrives at the ORB which transmits it to the appropriate POA. Subsequently, the object adapter starts the processing of the request by an implementation of the object (Servant).

The load management functionality, especially load monitoring and load distribution, have to be integrated into the ORB and the POA because we decided to make a system level implementation. Therefore, we added some policies and interfaces to the POA in order to enable state transfer and the creation of replicas. The monitoring of the runtime environment is performed via the Simple Network Management Protocol (SNMP) [11] which is a well-established standard in network management.

A new policy called `ControlFlowPolicy` that controls the creation and destruction of CORBA objects is added to the POA. The policy value `USER` indicates that objects are created by the programmer. The value `SYSTEM` indicates that objects are created on demand by the CORBA runtime environment. This enables the transparent creation of new objects in case of migration and replication. Therefore, the programmer has to provide a `ServantFactory` interface that enables the creation and destruction of Servants analogous to the Factory design pattern [4]. The POA's `RequestProcessingPolicy` is extended with the value `USE_SERVANT_FACTORY` that causes the POA to use the `ServantFactory` for object creation and destruction.

Migration and replication of objects that hold state require state transmission as described before. Therefore, some persistence mechanism has to be provided. A new policy, the `PersistencePolicy` is added to the POA. The policy value `USE_PERSISTENT_SERVANT_FACTORY` indicates that an extension of the `ServantFactory` interface, the `PersistentServantFactory`, is used in order to create and destroy objects. Additionally, the `PersistentServantFactory` provides the functionality to extract an object's state and to recreate objects from that state. This approach enables the application of various persistence mechanisms like the Persistent State Service [9] or proprietary mechanisms like Java serialization.

Finally, request redirection is performed by the CORBA Location Forward mechanism [5]. It enables to hand over object references to clients by raising an `ForwardRequest` exception. The client runtime transparently reconnects to the forwarded reference. This guarantees migration and replication transparency.

3. The medical image-processing application

A medical image-processing application is chosen for exploration of concept purposes. The realignment process forms part of the Statistical Parametric Mapping (SPM) application developed by the Wellcome Department of Cognitive Neurology in London [6]. SPM is used for processing and analyzing tomograph image sequences, as obtained, e.g., by functional Magnetic Resonance Imaging (fMRI) or Positron Emission Tomography (PET). Such image sequences are used in the field of neuroscience, for the analysis of activities in different regions of the human brain during cognitive and motoric exercises.

Realignment is a cost intensive computation performed during the preparation of raw image data for the forthcoming statistical evaluation. It computes a

4 × 4 transformation matrix for each image of the sequence, for compensating the effect of small movements of the patient, caused, e.g. by his breath. The images are realigned relatively to the first image of the sequence.

The realignment algorithm for image sequences as obtained by fMRI will briefly be presented. One has to distinguish two cases.

- *First case*: realignment of one sequence of images: The reference data set and the first matrix is obtained by performing a number of preparatory computations using the image data of the first image. The matrices for all remaining images are calculated using the reference data set.
- *Second case*: realignment of multiple sequences of images: The reference data set and the first matrix of the first sequence are calculated. Thereafter, the first images of all remaining sequences are realigned relatively to the first image of the first sequence and its reference data set. Finally, the realignment algorithm as described in the first case is applied to all sequences independently.

At this point the only precondition for the calculation of the transformation matrix is the availability of the reference data set, which is calculated only once for each sequence. Once the reference data set(s) is(are) available, the matrices of the sequence(s) can be computed independently.

The manually parallelized realignment application is already available as sequential C++, C++/CORBA and C++/PVM program. Previous work shows, that the overhead induced by CORBA is not prohibitive for its deployment in clinical environments.

For the following steps it is necessary to transform the sequential C++ program into a Java program because some components of our tool environment only provide Java interfaces. This program transformation is performed using the Java Native Interface (JNI). An interesting intermediate result is that the deployment of JNI does not lead to any performance decrease for the specific program [12].

4. Integrating the application into the tool environment

In order to improve performance and scalability of the image-processing application we decided to integrate it into our load management system.

As already mentioned in Section 3 the availability of a Java program is a necessary prerequisite for the integration into the load management system, since it only provides services for Java/CORBA programs. The sequential Java realignment application is transformed into a distributed Java/CORBA application.

Fig. 2 depicts the structure of the CORBA application. The service offered by the server object is the `compute( )` service, which calculates the transformation matrix for an image. The state of a server object consists of a reference data queue (cache). Therefore it is replication safe since it can be replicated without applying a consistency protocol to its replicas, i.e. the required cache data can easily be reestablished. A `getReferenceData( )` service is offered by each client and provides the specific reference data to the server if it is not already cached.

The basic adaptation of the Java/CORBA application to the load balancer is straightforward. Minor changes to the code are necessary in order to add the `ServantFactory` and `PersistantServant Factory` methods to the server object. In addition to those modifications the system is extended by various additional components for testing particular aspects of the load management system. The mechanism itself was integrated into the Java-based JacORB [1].

The second part of our tool environment consists of the Middleware Monitoring Tool (MIMO) [3] and

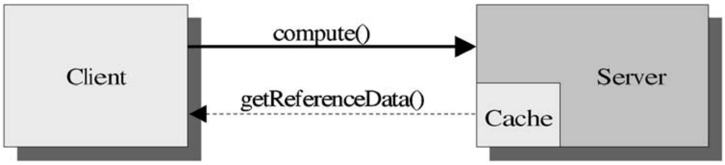

Fig. 2. The structure of the medical image-processing application.

the graphical on-line visualization tool MiVis (Middleware Visualization). The integration of these tools is straightforward too. MIMO provides some standard events like object creation, object deletion, object interactions, and additionally defines generic events. Furthermore, MIMO provides the infrastructure for designing active tools, i.e. tools that manipulate the monitored application. Initially, we specify the data to be monitored, for example client and server hosts, client and server objects, server object load, server host load, application object interactions, and load balancing actions like migration and replication. This information is provided by a MIMO adapter that is used to instrument the application and the load management system.

MiVis is a graphical on-line visualization tool that is based on the MIMO monitoring system. It provides a framework that enables the development of new display types which can be plugged into the tool core. We developed a new display that is used for the visualization of the new monitoring events described before. Fig. 3 presents the basic layout of the graphical on-line tool. Client and server objects are located within the respective rectangles representing the client and server hosts. In addition, server object load (numerical representation) and server host load values are depicted (numerical and graphical representation). The CORBA method `compute()` is represented as blue arrow (black in Fig. 3) with a counter and `getReferenceData()` as offset turquoise arrow. Replications and migrations are represented as yellow (white in Fig. 3) and red arrows, respectively. Replication and migration actions can be initiated manually too, by a drag and drop function.

The combination of MIMO and MiVis provides a flexible and extensible infrastructure for the development and the maintenance of large scale distributed applications. Together with our monitoring system performance and scalability of applications can be substantially improved.

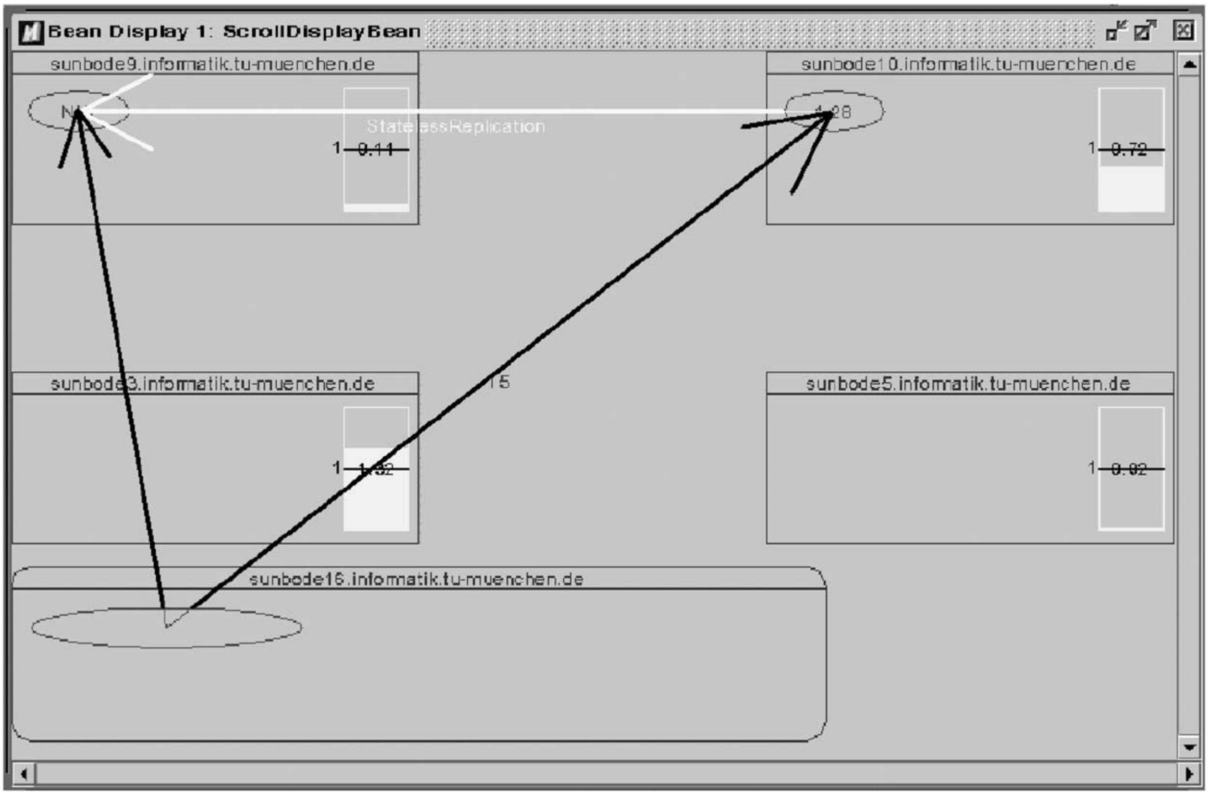
Bean Display 1: ScrollDisplayBean
sunbode9.informatik.tu-muenchen.de
N
StatelessReplication
1 0.11
sunbode10.informatik.tu-muenchen.de
1.28
1 0.72
sunbode3.informatik.tu-muenchen.de
15
1 1.32
sunbode5.informatik.tu-muenchen.de
1 0.02
sunbode16.informatik.tu-muenchen.de

Fig. 3. Visualization of a replication and of object interactions.

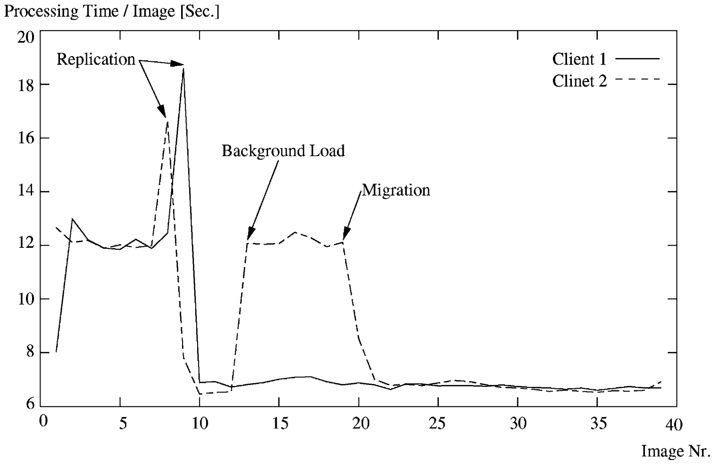

Fig. 4. The load managed medical image-processing application.

5. Evaluation

In order to evaluate the efficiency of the presented load management concept and its implementation, a test case is shown.

The hardware consists of three machines with equal configuration. There is no background load on the machines. The examined CORBA application is the medical image-processing application described in Section 3 with two simultaneously requesting clients. The application is replication safe as already mentioned in Section 4. Thus, migration and replication can be applied to this application.

Fig. 4 shows the processing time per image against the number of the processed image for both clients. At the beginning, one server object is created and placed on a machine (initial placement) and the clients start requesting the server. The image-processing time is equivalent for both clients now because the server alternately processes their requests. After a while the load management system recognizes that the server is overloaded because both clients permanently request the server. Accordingly, replication is performed, i.e. a second server object (replica) is created and each client gets a replica on its own. In consequence of the replication, the image-processing time of each client decreases about 50%. Some time later background processor load is generated on the machine that is used by the second client's replica. Hence, the image-processing time of the second client substantially increases. Again, the load management system recognizes the processor overload and migrates the affected replica to the third machine which was not used so far. The consequence is that the image-processing time returns to its normal level.

The test case shows how the load management system is able to deal with different kinds of overload. Request overload is compensated by replication, whereas background load is compensated by migrating an object to a less loaded host. Consequently, the load management systems improves the performance and the scalability of the medical image-processing application.

6. Conclusion and future work

The combination of load balancing and graphical user interface provides a powerful environment for the production oriented image-processing in medical environments. Workstation clusters can be used as high-performance servers for reconstruction and statistical analysis of tomography pictures. Our CORBA-based approach allows the integration of image-processing into the workflow of clinical routine. Future steps in this field will cover aspects of fault tolerance, where the computing environment will have integrated mechanisms for fail-soft and recovery.

Acknowledgements

This work is partly funded by the Deutsche Stifterverband, Kurt-Eberhard-Bode Stiftung.

References

[1] G. Brose, JacORB: implementation and design of a Java ORB, in: Proceedings of the International Conference on Distributed Applications and Interoperable Systems (DAIS'97), Chapman & Hall, London, 1997.

[2] G. Eddon, H. Eddon, Inside Distributed COM, Microsoft Press, 1998.

[3] G. Rackl, Monitoring and managing heterogeneous middleware, Ph.D. Thesis, Technische Universität München, Munich, Germany, 2001.

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley, Reading, MA, 1994.

[5] M. Henning, Binding, migration, and scalability in CORBA, Commun. ACM 40 (10) (1998) 62–71.

[6] K. Friston, SPM, Technical Report, The Wellcome Department of Cognitive Neurology, University College London, 1999.

[7] M. Lindermeier, Load management for distributed object-oriented environments, in: Proceedings of the International Symposium on Distributed Objects and Applications (DOA' 2000), Antwerp, Belgium, IEEE Press, New York, 2000.

[8] OMG (object management group), A discussion of the object management architecture, Technical Report, 1997. http://www.omg.org.

[9] OMG (object management group), CORBAServices: common object services specification, Technical Report, 1998. http://www.omg.org.

[10] OMG (object management group), The common object request broker: architecture and specification, Revision 2.3.1, Technical Report, 1999. http://www.omg.org.

[11] W. Stallings, SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, Addison-Wesley, Reading, MA, 1998.

[12] A. Stamatakis, Interoperable tool deployment for the late development phases of distributed object-oriented programs, Master's Thesis, Technische Universität München, Munich, Germany, 2001.

T. Ludwig received his diploma in computer science from Universität Erlangen, Germany in 1987. In 1988, he moved to the Technische Universität München where he worked on his PhD and afterwards as a research assistant. His main interest focusses on parallel computing. He conducts research in this field and teaches all aspects of parallel computing. He participated in many national and international projects in the field of parallel computing. His special interests are on-line tools for program development and specifically load balancing mechanisms. His current research is in the field of high-performance computing in bioinformatics.

A. Stamatakis studied computer science and aeronautical engineering as secondary subject and received his diploma degree in March 2001 from the Technische Universität München. He conducted internships and studies abroad at the Ecole Normale Superieure de Lyon, the National Technical University of Athens, the Eurocontrol Experimental Center (Paris), and the Instituto de Salud Carlos III (Madrid). In the past he has conducted work on advanced tools for air traffic control, load-balanced CORBA applications and monitoring tools, and data-bases for biochip experiments. Presently he is a PhD student at the Lehrstuhl für Rechnertechnik und Rechnerorganisation of the Technische Universität München and works in the field of high-performance bioinformatics.

M. Lindermeier graduated with a diploma in computer science from the Technische Universität München in 1998. Subsequently, he received a scholarship from the Technische Universität München to work on his doctoral thesis. Currently, he is affiliated with the Lehrstuhl für Rechnertechnik und Rechnerorganisation as doctoral researcher. His special field of interest is load management for distributed object-oriented environments.

G. Rackl received his PhD in computer science at Technische Universitat Munchen in 2001 with a thesis on monitoring and management of heterogenous middleware environments. Currently, he is engaged in the group IT management department of BMW Group in Munich, Germany, where he is dealing with tool support of the early phases of the software lifecycle. During his time at TU Munchen, he mainly participated in the ESPRIT project SEEDS (simulation environment for the evaluation of distributed traffic control systems) and in the SFB 342 research grant of the German Science Foundation (DFG), which dealt with tools for the development of parallel and distributed systems.