An Architecture for Automatic Load Distribution on Distributed Objects Computing Systems

Hermes Senger[1], Liria Matsumoto Sato[2]
[1] SENAC College of Computer Science and Technology
R. Galvão Bueno, 430 CEP 01505-000 São Paulo, SP - Brazil
{hermes.senger@cei.sp.senac.br}
[2] Polytechnic School of the University of Sao Paulo
Av. Prof. Luciano Gualberto, trav. 3, n. 180, São Paulo, SP - Brazil
{liria.sato@poli.usp.br}

Abstract—
In this paper we propose an architecture for load distribution in distributed object computing systems. Our strategy implements load distribution both at request level and at object level. The load distribution mechanism is integrated at service level, and is based on system resources monitoring and application monitoring. Finally, we discuss some implementation aspects and show that the architecture may be used in DOC systems such as CORBA, DCOM and Java/RMI.

Keywords— **Load management, load distribution, distributed object computing systems**

I. Introduction

Distributed middleware environments based on standardized protocols and network computing has become an interesting paradigm for parallel and distributed computing. As Distributed Object Computing (DOC) Systems became widely prevalent for building commercial, business and Internet based applications, many research interests have moved to improve such environments for general use, including scientific computation.

While scientific applications have traditionally claimed for high performance, commercial applications have recently experienced such demand, after large scale applications were created and deployed. Such demand for performance, together with low prices high performance desktops have created new challenges. New large applications, as well as legacy applications can be built and integrated into a great number of objects which can be placed in a single host, in a set of hosts or even worldwide.

In a distributed processing environment, the workload generated by one or more applications must be fairly distributed among the nodes for an efficient use of the available computing resources. Many factors can be a challenge for load distribution strategies. Processing nodes may present different hardware platforms which difficult service time prediction, different software platforms which may cause restrictions to load units distribution, and may be shared with local users and other applications which may lead to frequent load imbalance situations. Although earlier research on load distribution are based on mapping processes to processors [CAS88, EAG86], in the context of DOC systems the process is too coarse gained to be adopted as the basic load unit. The load distribution strategy must be request and object oriented, and must consider that objects may be shared by multiple client applications.

In this paper we present an architecture which implements mechanisms for automatic load distribution for general applications at service level, into a publishing component. Many applications can benefit from automatic load distribution facilities, such as farms of web servers (proxies, caches), application servers, large-scale simulation, movie rendering and others.

[1]Supported by FINEP/RECOPE, process number 3607/96.

II. Distributed Object Computing Systems

In this section we describe some basic concepts related to DOC systems and the project of load distribution strategies. In the sequence, we discuss some challenges to load distribution in DOC systems.

A. Basic Concepts

Load distribution has long been treated in the field of distributed operating systems [BAR93, SHI95]. In such approach, the process was frequently adopted as the basic load unit, so that load distribution was carried out by mapping processes to hosts [FER87]. In some situations a given running process could migrate to another host. Such operation requires the process to be blocked, its state be collected, sent to another host and resumed.

Some important requirements must be considered in the project of a load management mechanism. The middleware must provide *access transparency* to clients, regardless of the language the objects are implemented. *Location transparency* allows clients and servers to be completely unaware of the location of each other.

The *broker* mechanism which is based on the broker design patter [BUS96], depicted in figure 1 must provide support for transparent local and remote invocation, parameters

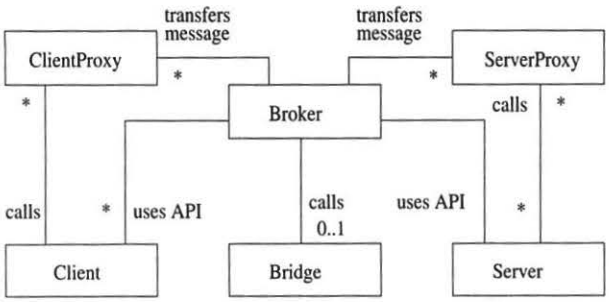

Fig. 1. The *broker* design pattern

passing and return of results. The broker usually implements a mechanism through which every object can be identified and found in the system, named *globally unique remote references*. *Proxies* may be used both at client and server side, to provide location transparency. Client side proxies are responsible for delivering method calls properly to server objects. Additionally, *bridges* may be used to provide communication over different middleware platforms using standard protocols.

Proxies may be *static* or *dynamic*. *Static proxies* always forward client requests to the same remote object, while *dynamic* proxies can forward subsequent calls to different objects. Dynamic proxies may implement dynamic request assignment for load distribution purposes.

DOC systems usually support some *persistence mechanism*, to collect and store an object state. It can be useful to implement migration and replication mechanisms. Additionally, mechanisms must be provided so that clients can dynamically discover object references. Such mechanisms may include the following publishing services: *naming services* which allow servers to register object references associated to *names* that can be queried by clients; *directory services* which support objects advertisement and discovery, based on entity attributes; and *trading services* [OMG00] that are also useful for objects advertising and discovery, based on their functionalities.

B. Load distribution principles

Many design principles are concerned with the project of a load management strategy for distributed object systems.

B.1 The load unit

DOC systems implement the object-oriented paradigm and client-server architecture which lead us to create strategies based on objects and requests. The process, commonly adopted as the basis for load management in earlier research, is too coarse grained because a single process may contain several objects in its address space. Migrating a process with several objects could be expensive and cause load imbalance situations.

Processes are too coarse grained to be adopted as the basic unit for load distribution because a single process may contain several objects. In DOC systems, a mix of load management at request level and at object level may lead to better results.

A server is an application process which can create and maintain service objects. In our architecture, servers can be classified in two categories.

Ordinary servers create and publish service objects in the distributed system in a very simple way. A server may also drop service objects down so that clients cannot use them anymore.

On demand servers do not create service objects previously. Instead, an on demand server publishes a list of supported services that can be started by creating objects on demand.

If more than a single server are registered for a given service, the load distribution component (see III.A.4), may choose one server according to its load situation and request.

B.2 Request level load distribution

If a service can be implemented by replicated objects, requests may be forwarded to any replica according to load distribution strategy. In order to implement replication, some earlier systems defined special types of objects. An example is the *volatile object* [GRI86], which does not hold a consistent state between calls. In our architecture, application level services may eventually be implemented by *replication safe* objects, which hold a special property. Lindermeier [LIN00] defines replication safe objects as: objects that can be replicated so that, at replication time, both the replica and the non-replicated object are equivalent.

Replication safe objects must not contain any references to other local objects in its internal state, because the granularity of replication is only one object. Therefore, the programmer must declare which objects are replication safe. After the replication, each replica has its own globally unique object reference in the system.

Clients make service requests against local proxies, which may forward such requests in two ways:

Static forwarding: chooses always the same object which implements the service. It is implemented by static proxies.

Dynamic forwarding: each request may be sent to a different object. Such approach is more fine grained, and may be implemented with dynamic proxies.

Thus, the forwarding policy is determined by the type of proxy distributed by publishing services to clients, as we describe in section III.A.2 .

B.3 Object level load distribution

Object level distribution distributes workload by placing objects in appropriate hosts, according to their capacity and load conditions. It can be done in three ways:

Initial placement: the load distribution mechanism chooses the location at which the object may be instantiated for the first time. In general, initial placement is limited to a set of registered on demand servers for service hosting.

Replication: may be used when the service is implemented by replication safe objects, and the existing replicas are itself overloaded or placed in overloaded hosts. Additionally, replication increases the number of alternatives for dynamic request forwarding in the future.

Migration: may be adopted for those replication unsafe objects, placed in overloaded servers. In this case, the state of the original instance must be captured and sent to a suitable target location. Migration may be carried out *preemptively*, calling the migration procedure immediately after the migration decision has been taken. A less expensive alternative is *non-preemptive migration*, in which the system lets the current calls finishing before proceeding with the migration and blocking only new incoming requests from running.

In general, replication is less expensive than migration because migration is a synchronous operation. The old instance only can be destructed after the new instance has successfully been created and the state has been transferred. In the opposite,replication can be carried out asynchronously by means of some lightweighted protocol.

As migration destructs the original instance and creates a new one, clients which have got references to the eliminated object must update their references. In our architecture, references updating functionality is supported by the trader component (see section III.A.2).

B.4 Load indexes

Several load indexes are imaginable to be used, in order to summarize the load situation of the system resources, such as: the *length of CPU queue*; the *CPU utilization rate*; the *response time* for services; and also, some *aggregated function* composed as linear combination of other elementary indexes. Additionally, several load indexes are imaginable to express utilization of other resources such as memory and network interface.

One of the most important results demonstrated in the bibliography [FER87, KUN91] shows that complex indexes do not necessary lead to significant performance improvements in the load distribution strategy. Ferrari and Zhou [FER87] demonstrated that load indexes based on a resource queue length are more effective than those based on its utilization rate.

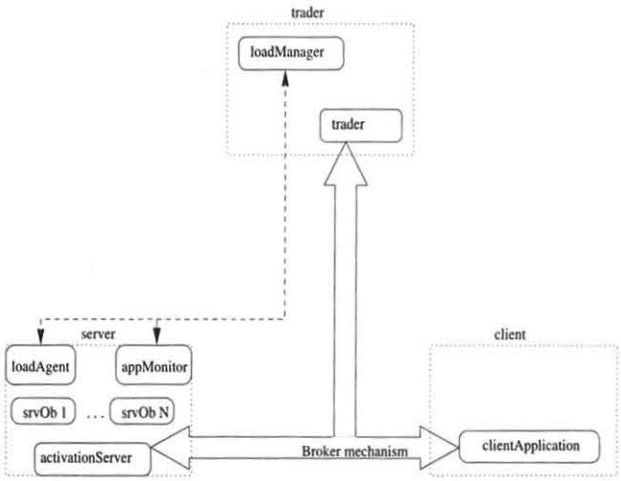

Fig. 2. The load management system architecture

III. An Architecture for Load Management

In this section, we describe a generic architecture for load management in distributed object systems, which covers the recommendations of the OMG *High Performance Working Group* about the minimum required functionalities for load management and aggregated computing [OMG01]: *system monitoring*, *information dissemination*, *performance evaluation*, *load distribution* and *load shading* for overload prevention.

A. *The components of the architecture*

This section describes the main components and functionalities implemented in the architecture, depicted in figure 2.

A.1 The load agent

The load agent is responsible for the *system monitoring* functionality at the server side. It gathers information such as the number of CPUs, amount of memory and swap space from the operating system kernel, and information about network resources from SNMP or RMON agents [STA97]. More than the static capacity parameters, it tracks the resources utilization dynamically. All the information are sent to the load manager component, described in III.A.4 .

A.2 The trader component

Some earlier works [SCH97b] have experienced the integration of load distribution strategies into a trader component, following the service level approach. The trader is an object that supports object advertising and discovery services within the system. Servers may advertise objects through the *export* operation, providing a description such as the *service*

type name, and the objects reference. Clients can invoke the *import* operation to discover objects which can match their needs. After the trader has passed object references, clients are able to make calls directly to remote objects.

A combination of both centralized and decentralized approaches can be implemented. Because the trader maintains load information received from the load agents, it can perform centralized load distribution procedures such as *initial placement*, *static request assignments*, *replication* and *migration* in a centralized fashion.

Additionally, the trader may delegate some load distribution decisions to clients, by returning a set of references or a dynamic reference (described in section III.B) for objects which implement the service. In this case, decentralized load distribution mechanisms require mechanisms to obtain load information at the client side. Such mechanisms can obtain load information from the load manager object (see section III.A.4) or directly from the load agents.

In order to provide additional scalability, the trader must support services and interfaces for other traders, so that clients located in different trader domains can interact. Additionally, if a bridge to other platform is provided, e.g. the General Inter-Orb Protocol (GIOP), traders may be implemented in different platforms.

A.3 The application monitor

System resources monitoring is a necessary but not sufficient condition for efficient load management. Application level monitoring and profiling must also be carried out. While system monitoring is concerned with the computing capacity offer, application monitoring is concerned with the demand for computing capacity, e.g. the total amount of CPU time used per request; the mean request rate for this server; the mean processing time for a request; the total amount of required memory; the size (in bytes) of a request, including parameters and returned results, and others.

Because application monitoring may consume computing and communication resources, it can be activated only during some initial period when a service is created or modified. As soon as the service requirements and behavior have been profiled, the application monitoring mechanism can be turned off. All the information about the application is sent to the trader and stored for load distribution decisions.

A.4 The load manager

The load manager implements part of the load distribution policy. It stores information received from the load agents, and provides summarized information to other components, such as:

- Which is the best service object for a given service, or a list of good service objects that can process requests from a client.
- Which is the best server or a list of good servers for initial placement and replication.
- Which is the best server or a list of good servers to be target of a migration.
- When a given server or the whole system can not receive new load units. This situation is named *shading condition* [OMG01], and prevents the system from overloading.

Such information are provided to the trader and clients (dynamic proxies) when proceeding to load distribution actions.

B. Dynamic proxies

Dynamic proxies can forward client requests to different objects along in time, according to the load distribution strategy. Dynamic requests forwarding can be useful in some situations, e.g. to discover new object references and provide transparent fault tolerance in case of server crash; for load balancing purposes, when services are implemented as replicated objects within the system; and to implement transparent redirection and references updating after object migration.

Dynamic proxies may work cooperativelly with *reference caches*, which provide information updates on which objects are still alive, information about load conditions and references updating after object migration.

C. The architecture implementation

We are currently implementing the proposed strategy on the top of the Java/RMI platform. Optionally, some agent platform can be used to implement migration more efficiently. Several refinements are involved in this project, such as algorithms which implement the load distribution strategy, policies for information dissemination such as polling or periodic dissemination, and general system tunning to assure low consume of computing and network resources. Such fine tunning process is strongly dependent of the timing imposed by the application, so that detailed monitoring and analysis activities must be carried out. Additionally, interfaces and policies for inter trader negotiation are under construction to allow additional scalability.

The load management architecture proposed can be implemented on the top any DOC platform that supports functionalities such as the broker mechanism; globally unique object references; and any kind of publishing service, e.g. a naming, directory or trading service where the load distribution strategy can be integrated. Therefore, our architecture can be implemented on the main middleware platforms such as CORBA, DCOM and Java/RMI.

Dynamic proxies are non trivial resource in DOC systems. If not supported, dynamic proxies can be implemented as a *wrapper* to the remote object, which implements the same

interface to the clients. In this case, tools for automatic generation of wrappers may be convenient. Dynamic proxies are being implemented with JINI resources.

IV. Related work

In the past, some works proposed load distribution mechanisms integrated into programming languages [COO80, JUL88]. Such approach does not deal with problems such as object sharing, which affects load distribution decisions. Furthermore it does not provide functionalities such as access transparency and location transparency.

Load distribution have also been studied under specific programming paradigms such as message passing [SCH97a], and in the field of distributed operating systems [BAR93, SHI95]. Although such approach has lead to many important results, most of them adopted the process as the load distribution unit, which lead us to develop new strategies and experiments in the context of DOC systems.

In the field of DOC systems, some works [BAR99] implemented load distribution mechanisms by extending the *naming service*. Despite of similarities to this work, only initial placement mechanism was used. Lindermeier [LIN00] propose the implementation of load distribution mechanisms at the system level, extending the CORBA standard. In our approach, load distribution operations are implemented at service level. Although the service level approach for load distribution does not facilitate the access to some system level information, it can lead to a less dependency of the resources offered by the middleware, and thus, it can be more generic.

V. Conclusion

In this paper, we present an architecture for load distribution in DOC systems. The service level approach have been chosen because it allows transparency to application programmers, and simplicity to the implementers because it does not interfere in the existing system internals. Part of the components and functionalities can be integrated into an existing publishing service such as a trading service, a naming service or directory service.

The architecture proposed implements load distribution at request level and at the object level. Request level load distribution tends to be less expensive because only requests are distributed among the existing replicas of a service object. Because it involves object creation, replication and migration, object level load distribution is more expensive and consequently must be used when request level load management is not possible. Additionally, request level load distribution can be carried out at the client side, leading to a more scalable system. Besides supporting system monitoring, the architecture must support application monitoring functionalities in order to be effective.

Our architecture covers all the functionalities OMG recommends for a load balancing and aggregated processing system. Additionally, it can be implemented on the top of broker mechanism, remote object identifiers, and some publishing service. If dynamic proxies are not supported, a similar solution can be implemented in order to support dynamic request assignments.

References

[BAR93] BARAK, A.; GUDAY, S.; WHEELER, R. G.; **The MOSIX distributed operating system: load balancing for UNIX**. LNCS, v. 672. Berlin, Springer, 1993.

[BAR99] BARTH, T.; FLENDER, G.; FREISLEBEN, B.;THILO, B. Load Distribution in a CORBA Environment. In: International Symposium on Distributed Objects and Applications (DOA'99). IN. **Proceedings International Symposium on Distributed Objects and Applications**, 1999.

[BUS96] BUSCHMANN, F;MEUNIER, R.; SOMMERLAND, P.; STAL, M; **Pattern-Oriented Software Architecture:A System of Patterns**. John Wiley & Sons, 1996.

[CAS88] CASAVANT, T. L.; KHUL, J.L. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. **Transactions on Software Engineering**. v.14, n.2, p.141–154. Feb, 1988.

[COO80] COOK, R. P. *MOD: a language for distributed programming. **Transactions on Software Engineering**. v.6, n.6, p.563–571. Nov, 1980.

[EAG86] EAGER, D. L; LAZOWSKA, E. D; ZAHORJAN, J. A Comparision of Receiver-Initiated and Sender-Iniciated Adaptative Load Sharing. **Performance Evaluation**, v.6, n.1, p. 53–68. Mar, 1986.

[FER87] FERRARI, D.; ZHOU, S. An Empirical Investigation of Load Indices for Load Balancing Applications. In: 12th Annual International Symposium on Computer Performance Modeling, Measurement and Evaluation. pp. 515-528. 1987.

[GRI86] GRIMSHAW A.S.; LIEP, J. W. S. Mentat: an object-oriented macro data-flow system. **ACM SIGPLAN Notices**, p.35–47, 1986.

[JUL88] JUL, E. Fine-grain mobility in the Emerald system. **ACM Transactions on Computer Systems**, v. 6, n.1, p.109–133. Feb, 1988.

[KUN91] KUNTZ, T. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme. **Transactions on Software Engineering**, v.17, n.7, p.725–730. Jul, 1991.

[LIN00] LINDERMEIER, M. Load Management for Distributed Object-Oriented Environments. In: Distributed Objects and Applications Symposium. Antuerp, Belgium, 2000. IN. **Proc. of the Distributed Objects and Applications. (DOA'00)**, p.50–68. IEEE Computer Society Press, 2000.

[OMG00] Object Management Group. **Trading Object Services Specification**. Object Management Group. Technical report. May, 2000.

[OMG01] Object Management Group. **Load Balancing and Performance Monitoring for CORBA-based Applications**. OMG Request for Proposal DRAFT (document ORBOS/2001-02-04). 2001.

[SCH97a] SCHNEKENBURGER, T.; STELLNER, G.(eds.). **Load Distribution for Parallel Applications**. Teubner, Germeany, 1997.

[SCH97b] SCHNEKENBURGER, T.; RACKL, G. Implementing Dynamic Load Distribution Strategies with Orbix. In: Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'97). IN. **Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications, 1997**, p.996–1006. Las Vegas, 1997.

[SHI95] SHIRAZI, B. A.; HURSON, A. R.; KAVI, K. M. **Scheduling and Load Balancing in Parallel and Distributed Systems.** Computer Society Press, Los Alamitos, CA, 1995.

[STA97] STALLINGS, W. **Snmp, Snmpv2 and Rmon**. Addison Wesley, 1997.