Symphony: Managing Virtual Servers in the Global Village*

Roy Friedman, Assaf Schuster, Ayal Itzkovitz, Eli Biham, Erez Hadad, Vladislav Kalinovsky, Sergey Kleyman, and Roman Vitenberg

Department of Computer Science
The Technion
Haifa 32000
Israel

Abstract. A *virtual server* is a server whose location in an internet is virtual; it may move from one physical site to another, and it may span a dynamically changing number of physical sites. In particular, during periods of high load, it may grow to new machines, while in other times it may shrink into a single host, and may even allow other virtual servers to run on the same host. This paper describes the design and architecture of *Symphony*, a management infrastructure for executing virtual servers in internet settings. This design is based on combining CORBA technology with group communication capabilities, for added reliability and fault tolerance.

1 Introduction

The emergence of internets, and in particular the Internet, created a potential for (literally) global resource sharing. If we examine the total load on all computing servers in the world at any given moment, we may see the following phenomena: While the load on a given computer may increase and decrease throughout the day, the overall load on all computers remains more or less constant. Moreover, at times when some servers are overloaded to the point of crushing, other machines are sitting idle.

To solve this problem, we propose the notion of *virtual servers*. A virtual server is a server whose identity is not bound to a fixed physical computer, but rather is changing and evolving with time and in reaction to the load on this server. Thus, a virtual server may move from one location in the network to another, and the number of physical computers on which it resides may change dynamically.

Consider, for example, an e-commerce application that shifts its location from the US to Japan when night falls on Los Angeles, and then from Japan to Israel when when day breaks in Tel-Aviv. In this example, being able to shift the location of the server guarantees that at any given moment, the server is located in proximity to its current clients. Another example is a large ray

* Supported by the Ministry of Science, Basic Infrastructure Fund, Project #9762.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 777–784, 1999.

tracing application running on a cluster of PCs in the Technion at night, using a distributed shared memory system like Millipede [8, 9]. This application suddenly notices that there are idle workstations at the Hebrew University in Jerusalem, and then decides to shift some of its computing tasks there, in order to enjoy higher parallelism. Similarly, Web based news sites become overloaded when an important event, such as elections or an important chess game, take place. These Web sites need to be able to grow, perhaps even to distant hosts, in order to handle the load.

In this work we describe the internal architecture of *Symphony*, an infrastructure for managing and executing virtual servers in internet settings.[1] Symphony's architecture is based on the fundamental principles underlying CORBA [2]. However, it is not a "pure CORBA" design, in the sense that services and applications (virtual servers) are replicated and can communicate internally in a non-CORBA compliant fashion. This distinction is important since it enables us to integrate high-performance servers, such as distributed shared memory, for which CORBA's performance is insufficient at best. Also, our main management services, as described later in this paper, are built using the Ensemble [7] group communication technology [3]; this design allows us to build scalable management services without giving up the power of group communication.

A service based design has the benefits of scalability, interoperability, and extendibility. The design is scalable, since the location of a service in the system is virtual. A service can run anywhere from a single host to the entire set of machines, yet be accessible everywhere. Also, services present an abstraction of objects, where only their interface is known, so that different implementation of the same service can be replaced without having to recompile the system. Finally, it is always possible to add services, or modify existing ones, in order to extend the functionality of the system.

In summary, our design tries to enjoy the benefits of both worlds. We use group communication as the main building block for our management infrastructure, use a CORBA-oriented architecture, and allow replicated/distributed servers to communicate internally with their own communication stacks for high performance. (Of course, servers that whish to communicate internally using CORBA can do so, but we do not impose this on other server.) We would like to emphasize that our work *does not restrict the programming model or language* (in particular we do not restrict ourselves to Java and/or sand-box models), yet puts emphasis on security, as discussed later in this paper.

2 Architectural Overview

Our architecture follows the CORBA approach of providing services to perform common tasks needed by applications and other services in the system. We arrange our services in several *domains*, as depicted in Figure 1. Services are *logically* organized in a hierarchical structure such that services in one domain

[1] A prototype of Symphony has recently been built, with an initial release planned for later this year.

tend to draw upon the functionality provided by services in lower or equal domains, but not in higher domains, as discussed below. Also, Symphony's run-time system is located below all services alongside with the the operating system.

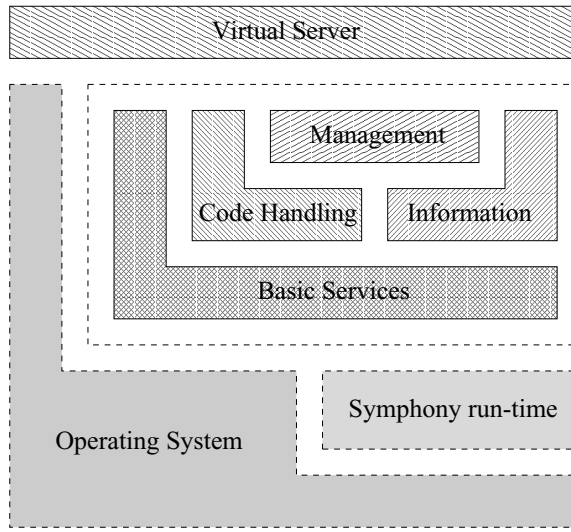

Fig. 1. Symphony's Service-Based Architecture.

Management services lie at the heart of our design. These services provide support for automatically replicating and migrating virtual servers based on the characteristics of these servers, the topology of the network, computers availability and so forth. The application can register with these services a description of its requirements and cost functions for locating "ideal" machines for it. The application is free to choose between two modes of operations: (a) The application can either allow the management services to automatically migrate and replicate itself based on the above descriptions (and restrictions), or (b) request non binding recommendations from the management services, but maintain the final word as to where and when its replicas should be (re)moved. This latter option is given for virtual servers that wish to retain control regarding the location of their replicas.

Management services draw upon the functionality provided by both *informational* services and *code-handler services*. The code-handler services do the actual task of starting an image of a code on a given machine upon demand, or evacuating machines under certain conditions. These services may receive instructions from either management services or applications, receive operational information from informational services, and utilize the functionality of the basic services as part of their normal operation.

Informational services, on the other hand, collect and report information about the system. This information includes the load on participating machines,

the topology of the system, the given environment on each of the machines, and the location of various applications. The information gathered by informational services can either be reported to subscribers using registered up-calls, or given as a response to an explicit request.

Basic services are located at the bottom of the hierarchy. These services provide the basic functionality that we view as necessary for most common applications, as well as to other services. These include a naming services, for registering and locating other services, a *security* service, *transaction* service, *replication* service, and an *event notification* service.

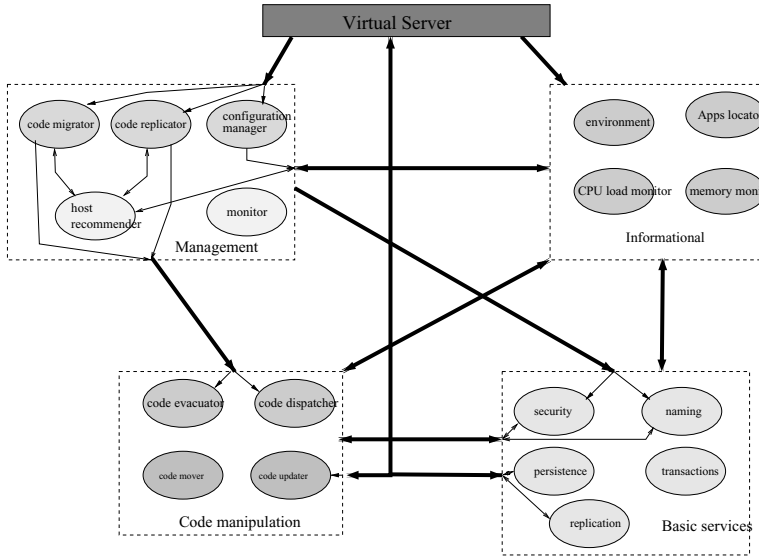

Fig. 2. Flow of information in Symphony.

Figure 2 presents a deeper look into the bowels of the system, by examining an ideal picture of the interactions between some of the main services in the system. In this example we see that the application (virtual server) registers itself with the management services, mainly the *code-replicator* and *code-migrator* services. (We discuss these services and others in the full version of this paper.) The code-replicator and code-migrator, in return, register with the informational services to be notified about the status of the system and to be notified about changes as they occur. When one of the management services decides to start a replica on a certain machine, or to evacuate the machine, it contacts the *code-dispatcher*, or *code-evacuator* respectively, to perform the job.

Applications as well as Symphony's services may use the name service and the security service to locate other services and to communicate securely with them.

2.1 Communicating with Replicated Services

Most of the services we propose need to be replicated for efficiency and high-availability. Also, different services may need different levels of replication. Some services may require a replica on every hosts, e.g., the code-dispatcher, while for other services it may be sufficient to instantiate a single replica in each cluster or subnet.

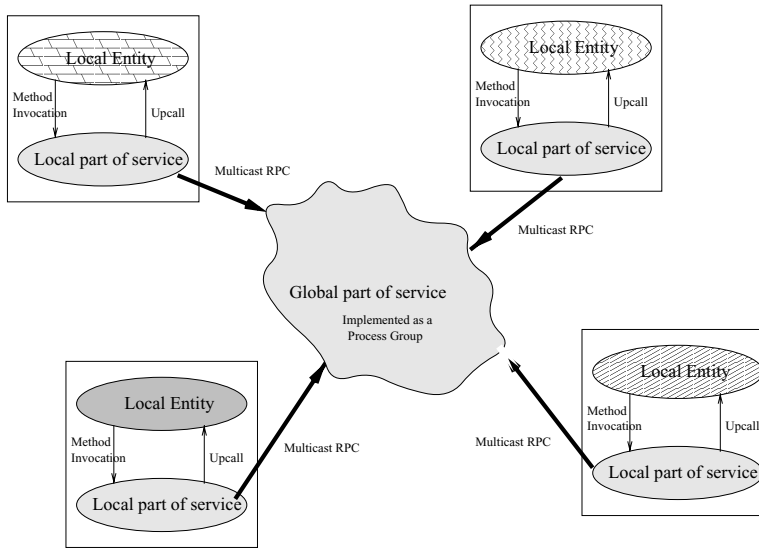

Fig. 3. Communicating with Replicated Services.

This goal is realized by splitting each service into a local stub that has to be started on each host, and a global part that is replicated, as illustrated in Figure 3. The global part consists of several replicas that form a process group; these replicas communicate through a group communication system that also manages membership changes, i.e., discovery of both failures of existing replicas and joins by new replicas.[2] The local stub communicates directly with the application or service entity on the local host. When contacted, the stub issues a multicast RPC request to the global part of the service, and is then responsible for collecting the replies and forwarding them to the local entity that initiated the request. In particular, the local stub can be asked to deliver the first reply that arrives from any replica, deliver all replies, or deliver the most common reply. Also, invocations on the local stub may be either synchronous, deferred synchronous, or asynchronous [10].

[2] The Event Notification Service of CORBA can be viewed as a plausible alternative for implementing replicated services. Felber and Guerraoui explain in [5] why ENS is in fact unsuitable for this task.

3 Security in Symphony

Trust Model A trust model, as suggested by its name, defines for each entity the kind of trust it has in other entities, and how this trust is verified. In Symphony, an entity can be a host, a message, or an image of a code. Each entity e holds a data structure that contains the following fields: (a) list of entities to whom e is willing to send messages, (b) list of entities from whom e is willing to accept messages, (c) list of entities to whom e is willing to send code, (d) list of entities from whom e is willing to accept code, (e) list of entities to whom e is willing to migrate/replicate, (f) list of entities to whom e is willing to provide services, and (g) list of entities with whom e is willing to share resources. Each of these categories is defined by specifying an access control list, and can be refined using the following subcriteria: (`i`) type (of communication/service/code), (`ii`) authentication scheme, (`iii`) limitations on the way the communication was carried, e.g., has to go through a firewall, (`iv`) other conditions for the interaction, and (`v`) recommendations in case multiple options occur.

Thus, it is possible to specify that host A is willing to accept any code written by Assaf, if it is authenticated using the El Gamal scheme, and it passed through a given firewall. A may also be willing to accept any Java applet that was generated at the Technion, and is signed with an MD5 hash function and a known secret key. Similarly, it is possible to specify that a replica of "*Your-Fastest-Web-Search-Engine*" never runs on the same machine as a replica of "*The-Most-Complete-Web-Search-Engine*", while some server A may require encryption of queries, but is willing to receive authenticated, but not encrypted, replies.

The trust model we have just described is not only used by virtual servers, but also by Symphony's internal services. In particular, some of the management and informational service may not accept certain types of data from untrusted entities. This is important in order to prevent malicious parties from creating havoc in the system. For example, if the host recommender recommends bad hosts, it may cause the system to migrate code to the wrong nodes, which will degrade the performance of the system, and make it unusable.

Message Security Securing messages can be done by authentication and encryption. Symphony supports both, and provides key management services for managing keys used in the entire system.

Securing Imported Code Execution In recent years there has been a large body of research on how to securely execute imported code. In Symphony, an exported code is accompanied with a *descriptive object*, which defines the restrictions the code is willing to run under, as well as the runtime environments it requires. Also, part of the information gathered by the informational services is the policies employed at each of the sites on the network towards such code. In particular, a code can be authenticated as coming from Assaf at the Technion, and some hosts in Jerusalem may be configured to allow full execution rights to Assaf's code, but restrict all other code to a Java-like sand-box model.

When sending code for remote execution, the code-dispatcher service consults with the appropriate information services regarding the possibility of placing the given code on the requested machine. The code is then executed on the remote site only if the site's policy allow the code to run there.

Moreover, the descriptive object accompanying the code includes the maximum amount of resources the process needs. Before starting a code image, the code-dispatcher verifies that the hosting machine is willing to provide that much resources to the imported code. These resources include memory consumption, disk space, cpu time, communication, and processes. If the answer is no, it will not be started on the given machine. If the answer is yes, it will be started, but the amount of resources it consumes will be monitored periodically; if they are exceeded, the process(es) will be stopped.

4 Related Work

Due to space limitations, here we only briefly mention some of the most related works. The full version of this paper includes a more detailed comparison of Symphony to other approaches and systems.

OSF DCE offers a service based environment for distributed computing [1]. DCE services include, e.g., a distributed time service, security, a distributed file system, and a distributed directory service. DCE aims to be a platform and vendor independent system, and supports its own RPC style communication. Symphony, on the other hand, concentrates on issues specific to virtual servers, and in particular supports services that automate the migration and replication of such virtual servers. Thus, Symphony could have been built as an extension over DCE, although we have chosen to use CORBA as our base platform.

Legion [6] is another object-oriented distributed system that provides an illusion of a single virtual machine composed of wide area collection of workstations, parallel computers, and fast massive multiprocessor machines. Class interfaces in Legion can be written in either CORBA's IDL or MDL [6]. Unlike Symphony, in which an application can choose the set of service it wises to use, Legion requires all objects to conform to its model. Each object should inherit from a predefines `LegionObject` and each class should be also instantiated and inherited from a base class named `LegionClass`. This programming model (rules) enables the Legion runtime system to efficiently support the mobility of application's objects, security in object invocation and heterogeneity. Legion deals with issues of scalability, security, fault-tolerance, and management of a large scale (sometimes called nation-wide) meta-computer. Note that, Legion address scalability by cloning objects, but does not take Symphony's approach of replicated services based on group communication.

There are several known approaches for combining CORBA and group communication, such as the ones developed in Electra and Orbix+ISIS [10]. The full version of this paper discusses them in more detail, and compare them to our approach.

User-level software-only distributed shared memory (DSM) systems, such as Millipede [8, 9], are another example of systems that need to migrate code in a cluster of workstations environment. In particular, in Millipede both code and data can migrate between machines based on their load and on access patterns to shared memory pages. Symphony complements Millipedes functionality, as Symphony provides services for replicating code, and for finding candidate machines for migrating code based on their load and communication capabilities.

Condor [4] and Utopia/LSF [11] are systems for doing batch scheduling in distributed environments. They focus on queuing many independent computation tasks and distributing them among a large network, whenever idle machines are detected. Since both these systems only handle independent computation tasks, they do not need to interact with the applications they run.

References

[1] DCE Home Page. http://www.osf.org/dce.
[2] The OMG Home Page. http://www.omg.org.
[3] K. Birman. The Process Group Approach to Reliable Distributed Computing. *Communications of the ACM*, 36(12):37–53, December 1993.
[4] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, and J. Pruyne. A Worldwide Flock of Condors: Load Sharing Among Workstation Clusters. *Journal on Future Generations of Computer Systems*, 12, 1996.
[5] P. Felber, R. Guerraoui, and A. Schiper. Replicating objects using the corba event service. In *Proc. of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems*, October 1997.
[6] A.S. Grimshaw and W.A. Wulf. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997.
[7] M. Hayden. The Ensemble Syste. Technical Report TR98-1662, Department of Computer Science, Cornell University, January 1998.
[8] A. Itzkovitz, A. Schuster, and L. Wolfovich. Supporting Multiple Programming Paradigm On Top Of A Single Virtual Parallel Machine. In *Proc. of Second International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'97)*, pages 25–34, April 1997. Earlier version appeared as Technion CS Technical Report LPCR #9607.
[9] A. Itzkovitz, A. Schuster, and L. Wolfovich. Thread Migration and its Applications in Distributed Shared Memory Systems. *The Journal of Systems and Software*, 1998. To appear. Also available as Technion CS Technical Report LPCR #9603.
[10] S. Landis and S. Maffeis. Building Reliable Distributed Systems with CORBA. *Theory and Practice of Object Systems*, April 1997.
[11] S. Zhou, J. Wang, X. Zheng, and P. Delisle. Utopia: A load sharing facility for large, heterogeneous distributed computing sytesm functionality. Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, 1992.