QoS-aware Clustering of Application Servers*

Giorgia Lodi, Fabio Panzieri
Dipartimento di Scienze dell'Informazione, Università di Bologna
Via Mura Anteo Zamboni 7, I-40127 Bologna (Italy)
email: lodig@cs.unibo.it, panzieri@cs.unibo.it

Abstract

In this extended abstract, we introduce a set of middleware services we are designing in order to enable clustered application servers to meet Quality of Service (QoS) application requirements. In addition, we summarize an implementation of these services we are currently carrying out as an extension of the JBoss application server, and discuss open issues concerning application deployment in clustered application servers.

1 Introduction

Current J2EE[16] -based application server technologies (e.g., JBoss [8], JOnAS [10], WebLogic [3], WebSphere [7]) support clustering of application server instances in order to host distributed, component-based applications. These technologies implement clustering solutions for scalability, load balancing, and fault tolerance purposes; however, they can meet only partially QoS requirements, such as availability, timeliness, security, and trust, of the applications they host as they are not fully instrumented for meeting those requirements (i.e., they are not designed to be *QoS-aware*, see below).

It is common industry practice to specify the QoS application requirements within a legally binding contract, termed Service Level Agreement (SLA), that includes both the specification of the QoS guarantees an application hosting environment has to provide its hosted applications with, and the metrics to assess the QoS delivered by that environment. The definition of SLAs is a complex task, and is outside the scope of this extended abstract (the interested reader can refer to [2, 18, 15]). For the purposes of our current discussion, we term *QoS-aware application hosting environment* a hosting environment designed to honour the so-called *hosting* SLAs; i.e., the SLAs that bind that environment to the applications it hosts.

*This research has been partly funded by the EU project TAPAS (IST-2001-34069) and the FIRB project entitled "WebMinds" of the Italian Ministry of Education, University and Research.

In order to construct one such an environment, the above mentioned SLAs are to be enforced, and monitored at run-time. Thus, we propose that SLA enforcement and monitoring be carried out by two principal middleware services, namely a *Configuration Service* (CS) and a *Monitoring Service* (MS), which can be incorporated in the current application server technology.

In addition, as an application hosting environment can be constructed out of a number of clustered application server instances, the CS and MS mentioned above are to be designed so as to exercise control over both the internal configuration of each server instance in that hosting environment, and the configuration of the cluster of server instances that form that environment (hence, in the following, we use the phrase *QoS-aware clustering* to refer to a QoS-aware hosting environment constructed out of clustered application servers).

In this extended abstract we introduce the aforementioned CS and MS, discuss their design in the context of application server clustering, and summarize their implementation. Typically, in this context, load balancing and fail-over mechanisms are essential in order to guarantee an effective clustering service. These mechanisms can be used by the CS and MS in order to govern the clustered resources and meet the hosting SLA. We are currently investigating the design of these mechanisms, and will not discuss them in this extended abstract.

Indeed, a number of research groups have investigated issues of resource clustering, and QoS enforcement and monitoring, both in the context of Web services, and application server technologies. Relevant results in this area, in addition to those already mentioned, include [14, 20, 17, 1, 6, 12, 5, 21]. For the sake of conciseness, we cannot compare and contrast our approach to QoS-aware clustering with those discussed in the cited references; however, we wish to mention that, to the best of our knowledge, none of the approaches proposed in the literature is based on aug-

menting existing application server technology with middleware services for SLA enforcement and runtime monitoring, as in our approach.

This extended abstract is structured as follows. In Section 2 we describe the design and implementation of our CS and MS. In Section 3 we discuss open issues concerning application deployment in clustered application servers. Finally, in Section 4, we introduce some concluding remarks.

2 Design and Implementation

The principal issues in the design of the CS and the MS can be summarized as follows. In general, the CS is responsible for configuring an application hosting environment (be this a single application server, or a cluster of servers) so that it meets effectively the SLAs that bind that environment to the hosted applications. To this end, the CS takes in input an application hosting SLA, and discovers the available system resources that can honour it. Provided that the discovered resources be sufficient to meet the input SLA, the CS reserves these resources and sets up the QoS-aware application hosting environment.

The MS is responsible for monitoring that hosting environment at application run time, so as to detect possible violations of the SLA. In order to prevent those violations, the MS is to be designed so that it takes appropriate actions if it discovers that an SLA violation is about to occur. Thus, for example, the MS can make use of a predefined *overload warning threshold* in order to detect dangerous load conditions that may lead to server overloading. In one such case, the MS can invoke the CS, and require that the application hosting environment be reconfigured appropriately, so that it adapts to the new load conditions, and continues to honour the application SLA.

Note that, in practice, the MS will have to make use of a number of warning thresholds (e.g., a throughput threshold, a response time threshold), which may contribute to define one such threshold as that mentioned above, for QoS monitoring purposes. These thresholds can typically be determined either by means of application benchmarking carried out prior to application deployment, for example, or by means of techniques based on the modeling and simulation of the hosting environment. In our Department, work is in progress on the design of some such techniques. However, this work is outside the scope of this extended abstract; hence, in the following, we will not discuss it further.

As the CS and the MS are responsible for configuring and monitoring both a single application server and a cluster of servers, they can be thought of as operating at two distinct levels of abstraction, that we term the *micro-resource* and the *macro-resource* levels, respectively. The former level consists of resources, such as server queues, and thread and connection pools, internal to each individual application server; the latter level consists of such resources as the group of clustered application servers, and their IP addresses.

Thus, for example, the CS at the micro-resource level is responsible for sizing appropriately an application server request queue, in order to enable that server to deal with an (anticipated) maximum number of concurrent requests, and to maintain its responsiveness. In contrast, in order to meet possible load balancing and responsiveness requirements, the CS at the macro-resource level may have to modify the cluster configuration at application run time, e.g., by enabling one (or more) new application server instance(s), or by replacing a crashed application server instance with an operational one.

The MS at the micro-resource level monitors the QoS (e.g., throughput, response time) delivered by the single application server, and requires the server reconfiguration when the delivered QoS reaches some predefined warning thresholds. At the macro-resource level, instead, the MS monitors the QoS delivered by the clustered application servers, and requires cluster reconfiguration in case the cluster delivered QoS reaches a predefined warning threshold.

The implementation of the CS and the MS at the Macro-resource Level, only, is discussed below.

2.1 Implementation

Our CS and MS services are being implemented as an extension of the JBoss application server. JBoss consists of a collection of middleware services for communication, persistence, transactions and security. These services interoperate by means of a microkernel termed *Java Management eXtension (JMX)* [19]. Specifically, JMX provides Java developers with a common software bus that allows them to integrate components such as modules, containers and plug-ins. These components are declared as *Managed Beans (MBeans) services*, which can be loaded into JBoss, and can be administered by the JMX software bus. The MBeans are the implementation of all the manageable resources in the JBoss server; they are represented by Java objects that expose interfaces consisting of methods to be used for invoking the MBeans.

A number of JBoss applications servers can be clustered in a network. A JBoss cluster consists of a set of nodes. A node, in JBoss, is a JBoss application server instance. There can be different clusters on the same network; each cluster is identified by an individual name. A node may belong to one or more clusters

(i.e., clusters may overlap).

A JBoss cluster can be used for either *homogeneous* or *heterogeneous* application deployment. Homogeneous deployment entails that each node in the cluster runs identical services, and Enterprise Java Beans (EJBs); in contrast, heterogeneous deployment entails that each node in the cluster may run a different set of services and EJBs. It is worth observing that, in practice, this latter form of clustering is not recommended [11]; hence, for the purposes of our current discussion, in the following we shall assume homogeneous deployment, only (issues of heterogeneous deployment are discussed further in Section 3). The JBoss Clustering service [13] is based on a framework consisting of a number of hierarchically structured services, and incorporating a reliable group communication mechanism, at its lowest level (currently implemented using JGroups [9]. The reliability properties of the JGroups protocols include lossless message transmission, message ordering, and atomicity).

The JBoss Clustering service implements load balancing of RMIs, and failover of crashed nodes (i.e., when a clustered JBoss node crashes, all the affected client calls are automatically redirected to another node in the cluster); these mechanisms are implemented inside the client stub.

Currently, client Java programs using the JBoss 3.2.x can choose among four load balancing policies (namely, Random Robin, Round Robin, First Available, and First Available Identical All Proxies). These policies implement non-adaptive load balancing within the cluster, at the RMI level. Hence, at run time, the chosen load balancing policy may select a target node in the cluster which may well be overloaded or close to being overloaded. This limitation cannot be overcome by these policies as they operate with no knowledge of the effective load of the clustered nodes, at run time.

However, JBoss allows its users to use, at the RMI level, additional user-defined load balancing strategies. Typically, these strategies can be designed so as to select the target nodes at run time, based on the actual computational load of those nodes. In addition, the latest JBoss version incorporates a so-called *HTTPLoadBalancer* Service which implements a response time based adaptive load balancing strategy, for HTTP sessions, only. This strategy is implemented at a higher level of abstraction than the RMI level mentioned above, and operates regardless of any hosting SLA.

Within this environment, we have created a new JBoss server configuration that provides deployed applications with the two middleware levels of our QoS-aware clustering service, i.e., the macro- and the micro-resource levels.

The macro-resource level is implemented by an MBean we have termed *MacroResourceManager*, which incorporates our CS and MS. This MBean uses the following auxiliary JMX services we have implemented in order to carry out the cluster configuration, reconfiguration and monitoring. The MacroResourceManager uses the following two MBeans: the *MeasurementService* MBean, which saves periodically the cluster state, and the *SLADeployer* MBean, which transforms the input SLA, specified in a XML form, into a Java object.

The MacroResourceManager implements our MS based on the monitoring architecture described in [15]. This implementation uses the above mentioned MeasurementService, and two specific classes termed *Macro Resource Monitoring* and *Evaluation and Violation Detection Service*.

The Evaluation and Violation Detection Service is responsible for monitoring, at run time, the adherence of the run time execution environment to the SLA; i.e., it detects whether the run time environment conditions (obtained from the Measurement Service) are close to violating the SLA, and decides the cluster reconfiguration strategy to be performed, if necessary.

The Macro Resource Monitoring is enabled by the MacroResourceManager, which starts the monitoring thread. This thread detects i) the current view of the cluster membership, ii) new members that join the cluster, iii) dead members that leave the cluster, and iv) the performance status of the cluster, in terms of throughput, response time, and probability of rejection parameters (note that these three parameters allow one to detect whether or not the nodes of the cluster are overloaded). In order to carry out its task, the Macro Resource Monitoring uses the JGroups communication interface available in JBoss, and the JBoss clustering framework.

Finally, the current implementation of the Macro Resource Monitoring sends periodically the data about the cluster membership, obtained from the JGroups framework, to the Measurement Service. This latter Service maintains these data in stable storage for logging purposes.

The CS in the MacroResourceManager MBean implements a distributed cluster configuration protocol, which can be summarized as follows. Assume that a JBoss cluster is set up, and that homogeneous application deployment is to be carried out within that cluster. Each JBoss node in that cluster embodies a CS instance in its own MacroResourceManager MBean;

this MBean is identified by a cluster-wide unique identifier (ID), assigned by the JGroups view management protocol.

In order to configure the application hosting environment, the actual application deployment is preceded by what we term an SLA deployment phase. In this phase, an SLADeployer is provided with an application SLA. This SLADeployer enables its local MacroResourceManager, which becomes the MacroResouceManager Leader of the cluster configuration. This Leader examines the input SLA, and contacts its peer MacroResourceManagers in the cluster in order to i) discover the resource availability at these MacroResourceManagers' nodes, and ii) construct a suitable cluster of nodes that can meet the input SLA. The possible crash of the Leader during a cluster configuration (or re-configuration) is detected via JGroups, and is dealt with by means of a simple recovery protocol which elects the MacroResourceManager with the currently smallest ID as the new Leader.

Note that the nodes in a cluster will host identical instances of the application, as homogeneous deployment is being carried out; hence, each node in that cluster should be capable of honouring the application SLA. As the cluster is started up, the actual application can be deployed and run. Clients can issue RMIs to any node in the cluster, transparently.

If a failure occurs, (e.g., the crash of a JBoss node in the cluster), the standard failover mechanism in JBoss redirects the client RMIs, addressed to the crashed node, to another active node in the cluster. In the standard JBoss clustering service, this node will be selected according to one of the four load balancing policies introduced earlier, and specified at deployment time. As these policies select a target node with no knowledge of the run time computational load of that node, it is possible that the RMI redirection following a node failure in a cluster lead to overloading another node in that cluster. In principle, this process may continue until all nodes in that cluster are brought to an overloaded state, as a sort of domino effect. (Note that this process may defeat the adaptive HTTPLoadBalancer as well).

In order to overcome this problem, in our implementation the CS aims to maintaining a fair distribution of the computational load among the clustered nodes. To this end, in case a node failure or an overload exception is raised by the MS within a cluster, our CS firstly attempts to reconfigure that cluster by integrating in it a spare node that replaces the faulty one; that spare node can be obtained possibly from another cluster (or from a pool of resources reserved for this purpose, for example). Secondly, if no spare node is available and the above reconfiguration cannot be carried out, the CS raises an exception to be dealt with at a higher level of abstraction (e.g., at the application level by adapting the application rather than the environment).

3 Open Issues

Issues of heterogeneous application deployment can play an important role in the implementation of our CS. In this Section, we introduce these issues, and examine three alternative implementations of our Service.

Heterogeneous deployment of application components consists of the ability of distributing the components of a single application across multiple clustered application servers, in a controlled manner. As already mentioned, this form of application deployment using the JBoss technology is not recommended, as there are a number of as yet unsolved problems related to it, including lack of i) distributed locking mechanisms and transaction management for use from entity beans, and ii) cluster-wide configuration management. However, we believe that heterogeneous deployment, in contrast with the homogeneous one discussed earlier, can be particularly attractive when applied to component-based technologies. Typically, these technologies adopt a distributed multi-tier paradigm, in which the application consists of separate components; namely, web components, and EJB components. In general, the Web components of an application are directly exposed to the clients, so as to mask the business tier in which the EJB components are located.

In this context, the clustered application servers can be specialized; thus, one application server instance can be configured for optimum performance for the support of transactions and Entity Beans, whereas another application server instance can be configured for optimum performance in the support of Session Beans. In addition, as we are considering a scenario in which client-server communications are enabled via wide area networks, as clients can be located geographically far away from the servers, it may well be convenient to distribute the application so that its Web components are as close as possible to the clients. Moreover, in order to reduce the application response times, it can be desirable to distribute the application EJB components so that those directly connected to the database (e.g. the Entity Beans) are located as close as possible to the clustered database servers. This can be done both at deployment time, when the CS acquires the application QoS requirements (i.e. the SLA), and at run time, when the application SLA is

close to being violated (as reported by the MS).

Owing to the above observations, three different implementation approaches of the CS suggest themselves; namely, a first approach implementing HEterogeneous Application Deployment (HEAD), a second one implementing a HOmogeneous Application Deployment (HOAD), and finally a third one implementing both forms of Deployment (HHAD). These three approaches are introduced below in isolation.

- HEAD: In order to implement heterogeneous deployment, the CS has to know in detail the Deployment Descriptors (DDs) of all the application components, at deployment time, so as to optimize the physical distribution of those components (i.e., if components communicate by means of local interfaces, they must be located in the same JVM, and are to be deployed so as to ensure that the SLA is met). At run time, it can be possible to migrate components from overloaded nodes to other, more lightly loaded, nodes (provided that local interfaces be maintained). The principal advantage of this approach is that the heterogeneous deployment allows the CS to distribute the computational load so as to optimise the use of the available resources in the cluster. In contrast, its principal shortcoming is its inherent complexity. Typically, migration of components requires that issues of state propagation, distributed locking, and management of a cluster-wide JNDI tree be carefully dealt with (this latter issue is addressed by the latest JBoss release); in addition, the application components DDs need to be examined in order to maintain the local interfaces.

- HOAD: If support for homogeneous deployment is required (i.e. copies of the entire application are located in every node of the cluster), the CS has to establish at SLA deployment time the most suitable cluster that can host the application. In this case, the entire application can be deployed in only one node of the cluster; then, the JBoss Farming service implements its distributed deployment. If the hosting environment conditions change at run time, and the SLA is about to being violated, the CS has to either choose a different cluster, or create a new one, and reorganize the application deployment. The implementation of this solution appears to be simpler than the HEAD solution discussed above, as it allows one to use the current JBoss clustering framework. However, in order to be effective, a HOAD solution requires that a new adaptive load balancing policy, which enables the clustered environment to meet the hosting SLA, be incorporated in the clustering mechanism at RMI level, as the currently available load balancing policies at this level suffer of the limitations discussed earlier.

- HHAD: Finally, the two approaches above can be combined as follows. At deployment time, the CS can execute homogeneous deployment by replicating component classes using the JBoss Farming service; thus, it will create a cluster in which the application can run, and load balancing applied. At run time, if the SLA is close to being violated, the re-configuration activity of the CS can decide whether to i) add more replicas of the entire application, ii) add a certain number of nodes to the initial cluster, i.e. to augment the number of available resources, iii) migrate application components from overloaded nodes either to other ones in the current cluster, or different nodes that do not belong to this cluster. The third option may lead to having different beans of the same application deployed and running onto different nodes (i.e., it may lead to heterogeneous application deployment). However, implementing this option may have a notable impact on the overall performance of the clustered application server, as migrating application components within the cluster can be very costly. Hence, this option can be used only when some particular hosting conditions occur (e.g. critical thresholds are reached within the hosting environment, and detected during the SLA monitoring phase). Finally, even in this third option, we can use the JBoss clustering features; however, some changes to the current JBoss implementation are required in order i) to apply an adaptive load balancing policy at RMI level, ii) to provide a distributed transactional manager, and iii) to both manage migration operations, and improve the performance of those operations.

To conclude this Section, we wish to point out that the implementation we have summarized in Section 2 of this extended abstract falls into the HOAD class of approaches introduced above; this approach has been chosen owing to its lower complexity than the HEAD and HHAD approaches. However, it is worth mentioning that we are currently developing implementations that fall into the HEAD and HHAD classes of approaches in order to compare and contrast the results obtained by different implementations.

4 Concluding Remarks

In this extended abstract we have discussed the design and implementation of a collection of middleware services we are developing in order to construct what we have termed a QoS-aware clustering service. This service is being developed as an extension of the JBoss clustering service.

It is worth observing that, in principle, a variety of independent applications can be hosted concurrently within the same physical cluster of resources. Each of these applications may have a different SLA with its hosting environment. In this context, it may well be worth using a cluster management policy that trades possible economic penalties, caused by the violation of a specific application SLA, for resources required by another application (e.g., as the penalties incurred in violating the SLA of the former application are less dramatic than those which would be incurred in by violating the SLA of the latter application). However, that cluster management policy may have to be based on a cluster-wide view of the state of the clustered resources. Depending on the cluster size, this approach may entail limitations in the scalability of the cluster management policy. We are planning to assess cluster management policies via experimental evaluation (see below).

Finally, we wish to mention that one of our principal research interests is in assessing those issues of homogeneous versus heterogeneous application deployment, discussed in Section3, in the context of a geographically distributed cluster of application servers. Specifically, we are investigating issues of QoS-aware geographical clustering across the Internet using a VPN technology, named VDE [4], which has been developed in our Department. This technology provides its users (i.e., the application servers, in our case) with the abstraction of an overlay Ethernet Local Area Network (LAN), constructed on top of the Internet. We are using this technology to build a VPN infrastructure where application servers can be clustered across the Internet as though they were connected to the same Ethernet LAN. The issues of application deployment discussed in Section 3, as well as the scalability issues mentioned above, will be investigated using this infrastructure.

Acknowledgements

We wish to thank our colleagues Graham Morgan (University of Newcastle upon Tyne), Andrea Ceccanti, and Gabriele D'Angelo (University of Bologna) for their useful comments on an earlier version of this extended abstract.

References

[1] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserve: A Mechanism for Resource Management in Cluster-based Network" *In Proc. ACM SIGMETRICS Conference*, Santa Clara, CA, June 2000.

[2] ASP Industry Consortium White Papers, SLA for Application Service Provisioning, http://www.allaboutasp.org.

[3] BEA, "BEA WebLogic Server 8.1 Overview: The Foundation for Enterprise Application Infrastructure", White Paper, August 2003.

[4] R. Davoli, "VDE: Virtual Distributed Ethernet", Technical Report TR UBLCS-2004-12, Department of Computer Science, The University of Bologna, June 2004.

[5] M. Debusmann, A. Keller, "SLA-driven Management of Distributed Systems using the Common Information Model" *In Proceedings of the 8th International IFIP/IEEE Symposium on Integrated Management (IM 2003)*, Colorado Springs, CO, USA, March 2003.

[6] http://www-106.ibm.com/developerworks/lotus

[7] http://www-306.ibm.com/software/webserver/appserv

[8] http://www.jboss.org

[9] http://www.jgroups.org/javagroupsnew/docs

[10] http://www.objectweb.org

[11] JBoss Group, "Feature Matrix: JBoss Clustering (Rabbit Hole)", 19 March, 2002.

[12] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services", IBM Reseach Report RC22456, March 2003.

[13] S. Labourey and B. Burke, "JBoss Clustering 2nd Edition", 2003.

[14] Miguel A. de Miguel, "QoS-Aware Component Frameworks" *In Proceedings of the 10th International Workshop on Quality of Service - IWQoS2002*, Florida, 2002.

[15] C. Molina-Jimenez, S. Shrivastava, J. Crowcroft and P. Gevros, "On the Monitoring of Contractual Service Level Agreements", *1st IEEE International Workshop on Electronic Contracting (WEC)*, July 2004, San Diego.

[16] B. Shannon, *Java 2 Plaftorm Enterprise Edition v. 1.4*, Sun Microsystem, Final Release 24 November 2003.

[17] K. Shen, H. Tang, T. Yang and L. Chu, "Integrated Resource Management for Cluster-based Internet Services" *In Proc. 5th Symposium on Operating Systems and Design and Implementation, USENIX Association*, Boston Massachussets, USA, December 9-11 2002.

[18] J. Skene, D. Lamanna and W. Emmerich "Precise Service Level Agreements" *in Proc. 26th of the International Conference on Software Engineering (ICSE'04)*, Edinburgh, Scotland (UK), 25 May 2004.

[19] Sun Microsystem, "Java Management eXtension: Instrumentation and Agent Specification v.1.1", 2002, http://java.sun.com/jmx.

[20] B. Urgaonkar, P. Shenoy and T. Roscoe, "Resource Overbooking and Application Profiling in Shared Hosting Platforms" *In Proc. 5th Symposium on Operating Systems and Design and Implementation, USENIX Association*, Boston Massachussets, USA, December 9-11 2002.

[21] T. Zhao and V. Karamcheti, "Enforcing Resource Sharing Agreements among Distributed Server Clusters" *In Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS)*, April 2002.