Autonomic administration of clustered J2EE applications

Sara Bouchenak[1], Noel De Palma[2], Daniel Hagimont[3]

[1] University of Grenoble I
Grenoble, France

[2] INPG
Grenoble, France

[3] INRIA
Grenoble, France

{Sara.Bouchenak, Noel.Depalma, Daniel.Hagimont}@inria.fr

1 Introduction

Today's computing environments are becoming increasingly sophisticated. They involve numerous complex software that cooperate in potentially large scale distributed environments. These software are developed with very heterogeneous programming models and their configuration facilities are generally proprietary. Therefore, the administration of these software (installation, configuration, tuning, repair …) is a much complex task which consumes a lot of resources:

- human resources as administrators have to react to events (such as failures) and have to reconfigure (repair) complex applications,
- hardware resources which are often reserved (and overbooked) to anticipate load peaks or failures.

A very promising approach to the above issue is to implement administration as an autonomic software. Such a software can be used to deploy and configure applications in a distributed environment. It can also monitor the environment and react to events such as failures or overloads and reconfigure applications accordingly and autonomously. The main advantages of this approach are:

- Providing a high-level support for deploying and configuring applications reduces errors and administrator's efforts.
- Autonomic administration allows the required reconfigurations to be performed without human intervention, thus saving administrator's time.
- Autonomic administration is a means to save hardware resources as resources can be allocated only when required (dynamically upon failure or load peak) instead of pre-allocated.

This paper presents Jade, an environment for developing autonomic administration software. Jade mainly relies on the following features:

- **A component model**. Jade models the administrated environment as a component-based software architecture which provides means to configure and reconfigure the environment. The same model is also used for developing the administration software itself.
- **A system representation** which provides a consistent and reliable view of the whole administrated system.
- **Control loops** which link probes to reconfiguration services and implement autonomic behaviors.

We implemented a first prototype of the Jade environment and used it for deployment and repair management of a clustered J2EE application.

In section 2, we introduce clustered J2EE applications. Section 3 presents an overview of Jade. Section 4 describes its application to the deployment and repair management of a J2EE server. We conclude the paper in section 5.

2 Clustered J2EE applications

2.1 J2EE applications

Java 2 Platform, Enterprise Edition (J2EE) defines a model for developing distributed applications in a multi-tiered architecture, e.g., e-commerce applications [7]. As illustrated by Figure 1, such applications are composed of the following tiers: Web, Servlet, EJB and Database. Upon an HTTP client request, either the request targets a static web document, in which case the web server directly returns that document to the client; or the request refers to a dynamic document, in which case the web server forwards the request to the servlet tier. A servlet is a Java program responsible for the Web page generation. The servlet may invoke EJB objects in order to compute data to be included in this page. Since EJBs may be persistent, they are backed up in the database.

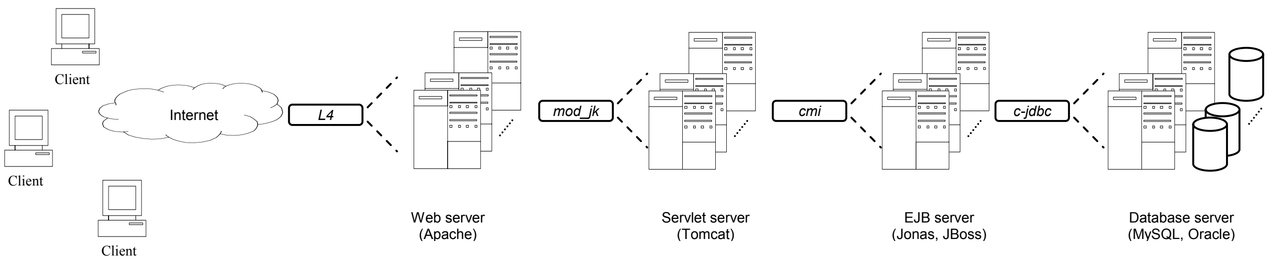

Figure 1. Clustered J2EE application

2.2 Clustering

To face high loads and provide higher scalability of J2EE applications, a commonly used approach is the replication of tiers on a cluster of machines. This approach defines a particular component in front of each tier, which dynamically balances the load among the replicas. Here, different load balancing algorithms may be used, e.g., Random, Round-Robin, etc. In addition to providing scalability, each load balancer also has the role of providing higher service availability. Indeed, it monitors liveness of the replicated servers and in case of a failure of a replica it simply ignores it and redirects the incoming requests to living servers. Among the existing clustering solutions, we can cite c-jdbc for a cluster of replicated database servers [3], cmi for a cluster of replicated Jonas EJB servers [4], mod_jk for a cluster of replicated Tomcat servlet servers [5], and the L4 switch for a cluster of replicated Apache Web servers [6] (see Figure 1).

2.3 Problem statement

In this context, the problem we are addressing is the administration of a clustered J2EE application.

First, the deployment and configuration of an application on a cluster is a much complex and time consuming task. We observed that settling such an application takes between one and two weeks for an experienced administrator.

Second, to provide scalability and availability, servers have to be replicated at the level of each tier. But when the replication policy has to be reconsidered (upon failure or server overload), a human intervention is required.

Our contribution with Jade is to provide an environment for developing autonomic administration software. Jade simplifies the configuration and deployment of a clustered J2EE application and it allows defining autonomic reconfiguration behaviors.

3 JADE

3.1 Component model

Jade relies on a component model, both for the administrated entities and for the implementation of the administration software. We used a Java-based and free open source implementation of a general component model called Fractal [2].

The use of a component model for building the administration software is motivated by the need to administrate the administration software itself, i.e. reconfigure it when required.

The administrated entities are encapsulated in components in order to allow their deployment and dynamic reconfiguration. The software resources that we encapsulated (as Fractal components) are all those involved in a J2EE multi-tier architecture. In our experiments, we chose free legacy software such as the Apache Web server, the Jakarta Tomcat servlet engine, the MySQL database server, and the RUBiS J2EE application that we used in our experiments (see section 4). Each encapsulated software provides its own implementation of Fractal's configuration interfaces. These interfaces allow managing components' attributes and in-coming and out-going bindings. Modifications of attributes or bindings are reflected on legacy software mechanisms. For instance, modification of the *port* attribute of the encapsulated Apache component is reflected in the *httpd.conf* file. Similarly, modification of out-going bindings of this Apache component is reflected in the *worker.properties* file.

Thus, reconfiguring a J2EE architecture consists in a component-based reconfiguration, instead of complex ad-hoc operation on legacy software configuration files.

3.2 Deployment

An application is described using an Architecture Description Language (ADL), which is one of the basic features of the Fractal component model. This description is an XML document which details the architectural structure of the application to deploy on the cluster: which software resources compose the multi-tier J2EE application, how many replicas are created for each tier, how are the tiers bound together, etc …

A *Software Resource Repository* component (a component of Jade) allows retrieving the encapsulated software resources involved in the multi-tier J2EE application (e.g., Apache Web server software, MySQL database server software, etc.).

A *Cluster Manager* component is responsible for the allocation of nodes (from a pool of available nodes) which will host the replicated servers of each tier.

The deployment of an application is the interpretation of an ADL description, using the *Software Resource repository* and the *Cluster Manager* to deploy application's components on nodes.

The autonomic administration software is also described using this ADL and deployed in the same way. However, this description of the administration software is separated from that of the application.

3.3 Self-repair

One important autonomic administration behavior we consider in Jade is self-repair[1]. Whenever a replicated tier fails, the service is still available thanks to replication. However, we aim at autonomously repairing the application by introducing a new replica in the application architecture in replacement of the faulting one.

In this purpose, Jade introduces three components: *System Monitoring*, *System Representation* and *Control Loop*.

The *System Monitoring* component monitors liveness of the cluster nodes through probes installed on nodes; those probes are implemented using heartbeat techniques.

The *System Representation* component maintains a representation of the current architectural structure of the application. Indeed, in order to tolerate a node crash, we need to manage a safe copy of the configuration of the components instantiated on that node (which would be lost otherwise). This representation is consistent in the sense that it reflects the current architectural structure of the application (which may evolve); and it is reliable in the sense that it is itself replicated to tolerate faults. The System Representation is implemented as a photography of the whole component architecture. This photography reproduces the whole architecture, but with empty encapsulated components (e.g. Apache, Tomcat …). It only captures the attributes and bindings of components in order to allow repairs.

The *Control Loop* component implements the autonomic behaviour, which is the repair management in our case. It receives notifications from the *System Monitoring* component and upon a node failure, makes use of the *System Representation* to retrieve the necessary information about the failed node (i.e., software resources that were running on that node prior to the failure and their bindings to other resources), contacts the *Cluster Manager* to allocate a new available node, contacts the *Software Resource Repository* to retrieve the necessary software resources, redeploys those software resources on the new node. In the particular case of the clustered J2EE application, the algorithm acts as follows:

- Say NC is the component deployed in replacement of OC
- Configure NC's out-going bindings (the same as OC)
- Start NC
- Configure the in-coming bindings (for each component X that was bound to OC)
- Restart each component X (generally required to take into account bindings updates)

The *System Representation* is then updated according to this new configuration.

Notice that different Monitoring Systems or Control Loops can be implemented according to the autonomic administration services that have to be defined. In particular, Control Loops can be defined to repair any faulting component of Jade (including Monitoring Systems or Control Loops themselves).

4 Experiments

In the following, we consider an experimental scenario that exhibits self-repair of a muti-tier J2EE application, the RUBiS auction site (modeling *eBay*-like sites) [1]. Here, we consider a multi-tier J2EE architecture consisting of a one Apache server as the Web tier, four Tomcat servers as the servlet tier, and one MySQL database server as the database tier. Mod_jk is used as a module (i.e., a library) interfacing the Apache Web server with the Tomcat servlet servers. All the used software systems, in addition to the RUBiS J2EE application, were deployed on the J2EE architecture using the JADE deployment mechanism (section 3.2).

[1] But many different behaviors may be defined.

Deployment

This application architecture is described using Fractal's ADL. The ADL description is given in Figure 2.

```
<definition name="fr.jade.test.J2EE">
...
<component name="apache"
definition="fr.jade.resources.ApacheResource">
  <attributes
  signature="fr.jade.api.GenericAttributeController">
  <attribute                         name="resourceName"
value="apache"/>
  <attribute name="dirLocal" value="/tmp/apache"/>
  <attribute name="user" value="depalma"/>
  <attribute name="group" value="sardes"/>
  <attribute name="port" value="8080"/>
  <attribute name="jkMounts" value="servlet"/>
  </attributes>
  <virtual-node name="node1"/>
</component>

<component name="tomcat1"   ... </component>
<component name="tomcat2"   ... </component>
<component name="tomcat3"   ... </component>
<component name="tomcat4"   ... </component>
<component name="mysql"  ...  </component>
...
<binding client="apache.out0" server="tomcat1.in"/>
<binding client="apache.out1" server="tomcat2.in"/>
<binding client="apache.out2" server="tomcat3.in"/>
<binding client="apache.out3" server="tomcat4.in"/>
<binding client="tomcat1.out" server="mysql.in"/>
<binding client="tomcat2.out" server="mysql.in"/>
<binding client="tomcat3.out" server="mysql.in"/>
<binding client="tomcat4.out" server="mysql.in"/>
</definition>
```

Figure 2. ADL description of the J2EE application

Each component creation includes the initialization of key attributes which are reflected in the encapsulated software configuration files before the software are launched (for instance, the *dirLocal* attribute of each component indicates the place where the software should be installed on the local host). At the end of this ADL description, the bindings between the created components are defined.

The high level of abstraction of this definition and the automatic deployment of the application, reduce configuration errors and simplify the task of administrators.

Moreover, the application being structured in terms of Fractal components, it benefits from all the interfaces defined in Fractal for managing components, especially managing component attributes and bindings. The repair procedure relies on these interfaces to update the application architecture.

Repair

In this experimental scenario, the deployed J2EE application is victim of 2 successive failures, respectively affecting two nodes hosting a Tomcat servlet server. When the *System Monitoring* component detects a failure, an event is delivered to the *Control Loop*. The *Control Loop* reallocates a new node and redeploys the necessary software resources on that node (i.e., a servlet server) and rebinds the involved components. The implementation of this reconfiguration in the *Control Loop* component is quite simple. It exploits the Fractal interfaces to:

- inspect the software architecture and determine what has to be repaired. This inspection is made on the *System Representation*, which is a (reliable) photography of the application (before failure).
- reconfigure the actual application architecture and replace the faulting component.

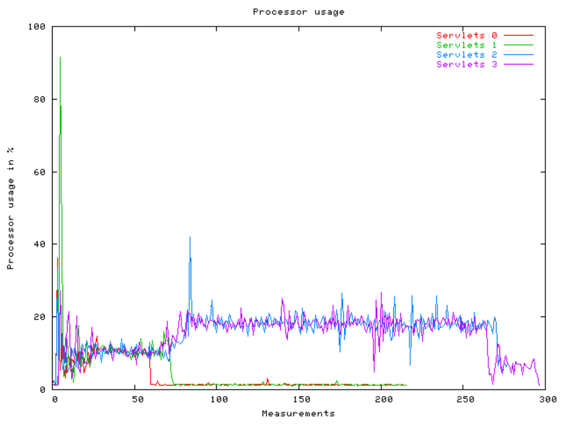

Figure 3. CPU usage on Tomcat nodes (without Jade)

Figure 3 shows the evolution of CPU usage on each Tomcat node, without repair by the Jade administration tool. In order to simulate the failures, we simply killed the Tomcat servers on the relevant machines. We observe that the CPU usage on the faulting machines falls down to zero, and that the CPU usage on the two remaining Tomcat servers increases consequently, as the mod_jk connection between Apache and Tomcat distributes the load among the Tomcat servers which are still alive.

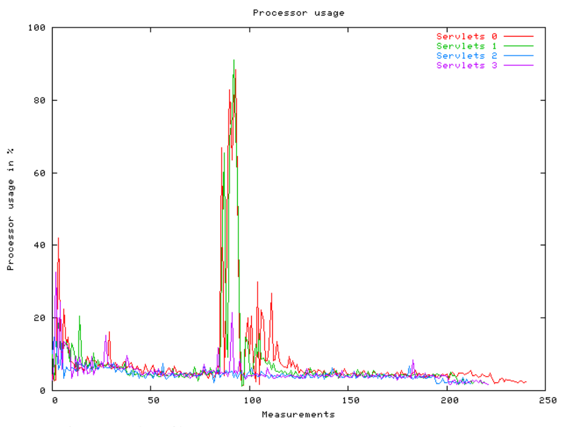

Figure 4. CPU usage on Tomcat nodes (with Jade)

Figure 4 shows the evolution of CPU usage on each Tomcat node, with the Jade repair policy previously described. For these measurements, the faulting machines are returned to the pool of free nodes and reallocated by the repair procedure. Therefore, the Tomcat servers are restarted on the same nodes. We observe a CPU peak on the faulting machines due to server restarts. After the repair period, the CPU level stabilizes at the same level as before the failures.

5 Summary and future work

This paper is a summary of our on-going work on Jade, an environment for implementing autonomic administration software.

The main motivation of this project is to deal with the difficult issues that administrators have to face in today's distributed computing environments. The important issues we identified are:

- Reduce the complexity of configuration and deployment
- Limit human intervention for administrating software

We address these issues in Jade with:

- A component model which provides a high level of abstraction for deploying, configuring and reconfiguring the administrated software.
- Autonomic administration which can be used to autonomously administrate software without human intervention.

Our experiments show the potential of this approach.

We are currently experimenting with more elaborated autonomic administration services which aim at repairing a wider spectrum of failures (software faults, network partitions …). We are also exploring the definition of an autonomic QoS manager which would be able to dynamically reconfigure the application (e.g. the number of replicas in a clustered J2EE application) according to the observed load.

Acknowledgment. Special thanks to Adrian Mos for the first measurements on Jade. This work has been partly supported by the European Community through the CoreGrid Network of Excellence.

6 References

[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani and W. Zwaenepoel. Specification and Implementation of Dynamic Web Site Benchmarks. *IEEE 5th Annual Workshop on Workload Characterization (WWC-5)*, Austin, TX, USA, Nov. 2002. http://rubis.objectweb.org

[2] E. Bruneton, T. Coupaye, and J.B. Stefani. *Recursive and Dynamic Software Composition with Sharing*. Seventh International Workshop on Component-Oriented Programming (WCOP02), Monday, June 10, 2002, Malaga, Spain. http://fractal.objectweb.org/

[3] E. Cecchet, J. Marguerite, W. Zwaenepoel. C-JDBC: Flexible Database Clustering Middleware. *FREENIX Technical Sessions, USENIX Annual Technical Conference*, Boston, MA, Etats-Unis, jui. 2004. http://c-jdbc.objectweb.org/

[4] JOnAS Project. *Java Open Application Server (JOnAS): A J2EE Platform*. http://jonas.objectweb.org/current/doc/JOnASWP.html

[5] G. Shachor. *Tomcat Documentation. The Apache Jakarta Project*. http://jakarta.apache.org/tomcat/tomcat-3.3-doc/

[6] S. Sudarshan, R. Piyush. *Link Level Load Balancing and Fault Tolerance in NetWare 6*. NetWare Cool Solutions Article, mar. 2002. http://developer.novell.com/research/appnotes/2002/march/03/a020303.pdf

[7] Sun Microsystems. *Java 2 Platform Enterprise Edition (J2EE)*. http://java.sun.com/j2ee/