# Self-Sizing of Clustered Databases

Christophe Taton[1], Sara Bouchenak[2], Noël De Palma[1], Daniel Hagimont[3], Sylvain Sicard[2]
*(1) Institut National Polytechnique de Grenoble, France*
*(2) Université Joseph Fourier, Grenoble, France*
*(3) Institut National Polytechnique de Toulouse, France*
*{Christophe.Taton, Sara.Bouchenak, Noel.Depalma, Sylvain.Sicard}@inria.fr*

## Abstract

*Distributed software environments are increasingly difficult to manage. This paper presents a middleware for the development of self-manageable and autonomic systems. Preliminary experiments for automatically adapting a cluster of replicated databases according to QoS requirements are reported.*

## 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.
- **Control loops** which link probes to reconfiguration services and implement autonomic behaviors.

We used Jade to implement self-sizing in a cluster of replicated databases. Here, self-sizing consists in dynamically increasing or decreasing the number of database replica in order to accommodate load peaks.

The remainder of the paper is organized as follows. Section 2 presents an overview of the Jade middleware for the implementation of autonomic systems. Section 3 describes our experiments with Jade for clustered databases self-sizing. Finally, Section 4 reviews related works and Section 5 draws our conclusions.

## 2. The Jade middleware

Throughout this paper, we illustrate the use of Jade with the management of a clustered J2EE application which is composed of several distributed interconnected components: Apache Web servers, Tomcat Servlet engines and MySQL database servers. The experiments described in Section 3 focus on the database tier of such applications.

### 2.1. Component-based management

We propose to use a component model as a base for the design and implementation of autonomic software.

The component model that we use is Fractal [4], which has the following benefits:

- it defines a hierarchical composition model for components, allowing complex architectures to be built;
- it provides a uniform, adaptable control interface that allows introspection and reconfiguration of component architectures.

The Fractal component model is used to implement a management layer for a set of (legacy) hardware/software elements. An element, or set of elements, is wrapped in a Fractal component. This provides a means to:

- Managing legacy entities using a uniform model (the Fractal control interface), instead of relying on element-specific, hand-managed, configuration files.
- Managing complex environments with different points of view. For instance, using appropriate composite components, it is possible to represent the network topology or the configuration of a clustered J2EE middleware (a distributed software architecture).
- Adding a control behavior to the modelized legacy entities (e.g. monitoring, interception and reconfiguration).

In the management layer, all components provide the same (uniform) management interface for the encapsulated elements, and the corresponding implementation (the wrapper) is specific to each element (e.g. in the case of J2EE, Apache web server, Tomcat Servlet server, MySQL database server, etc.). The interface allows managing the element's attributes, bindings and lifecycle.

Relying on this management layer, sophisticated administration programs can be implemented, without having to deal with complex, proprietary configuration interfaces, which are hidden in the wrappers. The management layer provides all the facilities required to implement such administration programs:

- Introspection. The framework provides an introspection interface that allows observing the components. For instance, an administration program can inspect an Apache web server component (encapsulating the Apache server) to discover that this server runs on node1:port 80 and is bound to a Tomcat Servlet server running on node2:port 66. It can also inspect the overall J2EE infrastructure to discover that it is composed of two Apache servers interconnected with two Tomcat servers connected to the same MySQL database server.
- Reconfiguration. The framework provides a reconfiguration interface that allows control over the component architecture. In particular, this control interface allows changing component attributes or bindings between components. These configuration changes are reflected onto the legacy layer. For instance, an administration program can add or remove an Apache replica in the J2EE infrastructure to adapt to workload variations.

As soon as wrappers have been implemented for legacy elements, any administration program relies on this uniform component model.

## 2.2. Control loops

An autonomic software is based on a control loop with the following components:

- First, *sensors* that are responsible for the detection of the occurrence of a particular event, e.g. a database failure, or a QoS requirement violation.
- Second, *analysis/decision* components that represent the actual reconfiguration algorithm, e.g. replacing a failed database by a new one, or increasing the number of resources in a cluster of replicated databases upon high load.
- Finally, *actuators* that represent the individual mechanisms necessary to implement reconfiguration, e.g. allocation of a new node for a cluster.

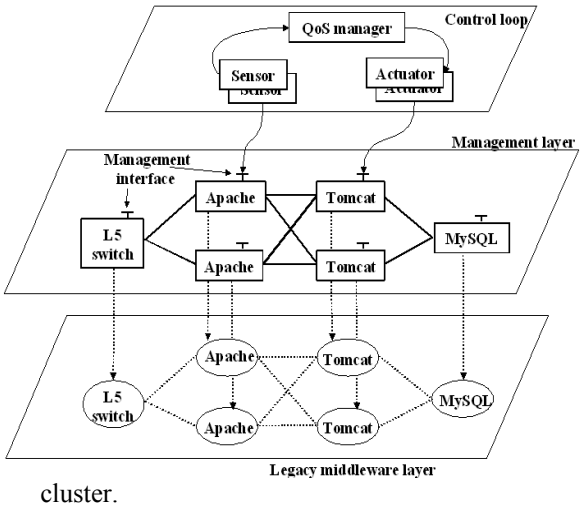


**Figure 1. Architecture of Jade**

The above approach is illustrated in Figure 1 in the case of a J2EE architecture. In this setting, an L5-switch balances the requests between two Apache server replicas. The Apache servers are connected to two Tomcat server replicas. The Tomcat servers are both connected to the same MySQL server. The vertical dashed arrows (between the management and legacy layers) represent management relationships

between components and the wrapped software entities. In the legacy layer, the dashed lines represent relationships (or bindings) between legacy entities, whose implementations are proprietary. These bindings are represented in the management layer by (Fractal) component bindings (full lines in the figure). At the top, control loops can be implemented, relying on introspection and reconfiguration interfaces of the Fractal component model.

# 3. Experimental evaluation

## 3.1. Self-Sizeable Databases

A standard pattern for resource allocation in clustered servers is a load balancer which distributes incoming requests among a set of replicated servers. The distribution algorithm is usually Round-robin (equally distributing the load between servers). Generally, the number of server replicas is statically defined.

We used Jade to autonomously adjust the number of replicated database servers in a clustered J2EE application when the load varies. The expected benefits are (i) improving resource utilization; and (ii) preserving user-perceived performance in the face of wide variations of the load.

The control loop algorithm is based on two thresholds related to the CPU load. These thresholds are tuned so that they trigger reconfigurations when the resources are effectively overloaded or underloaded.

Database replication relies on C-JDBC [5], a load balancer dedicated to the replication of database back-ends. Each replica contains a full copy of the database (full mirroring). C-JDBC parallelizes read-only requests among the replicas. Write requests are applied to all replicas so that each running copy of the database remains up-to-date. This (read one, write all) strategy is applied by the C-JDBC load-balancer on a static set of database back ends.

We wrapped the C-JDBC load-balancer in a Fractal component to enable it to be managed by the Jade framework. Notably, this C-JDBC component allows addition and removal of database back-ends.

To manage a dynamic set of database back ends, a newly allocated back-end must synchronize its state with respect to the whole clustered database before it is activated. To do so, a "recovery log" has been added to the C-JDBC load-balancer. This recovery log is implemented as a simple database whose purpose is to keep trace of all the requests that affect the state of the back-ends. Basically, all write requests are logged and indexed as strings in this recovery log. When a new back-end is inserted in the clustered database, the state of this back-end is potentially not up-to-date. The recovery log enables us to know the exact set of write requests to replay on this back-end to make it be up-to-date. Once these requests have been processed by the newly allocated back-end, we can reinsert it in the clustered database as an active and up-to-date replica.

## 3.1. Evaluation environment

Initially, the J2EE system is deployed with one application server (Tomcat) and one database back-end (MySQL). The deployed application is RUBiS[1], a J2EE application benchmark based on servlets, which implements an auction site modelled over eBay. The following workload has been submitted to our managed J2EE system: (i) at the beginning of the experiment, a medium workload is submitted: 80 emulated clients; then (ii) the load increases progressively up to 500 emulated clients: 21 new emulated clients every minute; finally (iii) the load decreases symmetrically down to the initial load (80 clients).

The control loop algorithm is executed every second. This time interval is short enough to quickly detect performance variations and to react promptly. In order to have a consistent load indicator, the CPU usage is smoothed by a temporal average (moving average). The strength of this average is experimentally fixed accordingly to the variability of the CPU usage observed during benchmarking experiments. For instance, the average CPU usage is computed over the last 60 seconds for the application servers and over the last 90 seconds for the database back-ends. The experimental evaluation has been performed on a cluster of x86-compatible machines (Processor Intel Xeon 1800 Mhz, 1Gb RAM). The nodes are connected through a 100Mbps Ethernet LAN to form a cluster. The experiments required up to 9 machines: one node for the Jade management platform, one node for the frontal load-balancer (L5-switch), up to two nodes for the web (Apache) and servlet (Tomcat) servers, one node for the database load-balancer (C-JDBC), up to three nodes for the replicated database back-ends (MySQL), and finally one node for the client emulator (which emulates up to 500 clients).

## 3.2. Experimental results

Figure 2 shows the effect of the control loop on the number of database replica.

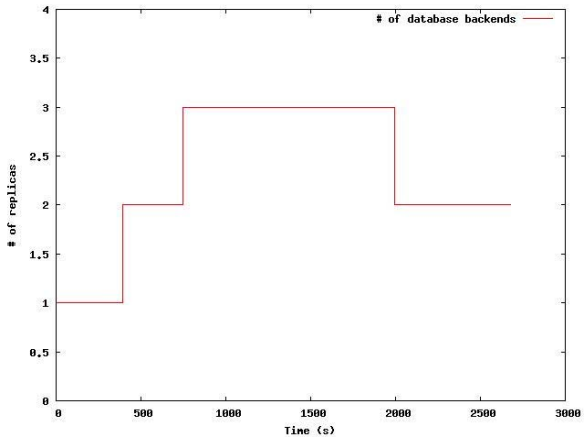


**Figure 2 Dynamically adjusted number of replicas**

This behavior may be explained as follows.

- As the workload progressively increases, the average resource consumption of the clustered database also increases, until the CPU threshold is reached (for about 180 clients), which triggers the allocation of one new database back-end. The system now contains two database back-ends.
- The workload continues to grow, and triggers a second node allocation (for about 320 clients) for the clustered database. The system is here composed three database back-ends.
- The workload then increases (up to 500 clients) without saturating this database configuration, and then starts decreasing.
- The workload decrease implies a decrease of the resource consumption of the clustered database, which triggers (for about 280 clients) the deallocation of one database back-end.

To quantify the effect of the reconfigurations, this scenario (the workload) has also been experimented without Jade, i.e. without any reconfiguration, so that the managed system is not resized. Figure 3 reports the results of these experiments and shows the thresholds used to trigger dynamic reconfigurations (the curve of Figure 2 is also reported on Figure 3). When the average CPU usage reaches the maximum threshold set for the database, the control loop triggers the deployment of a new database back-end, which implies a decrease of the average CPU usage. Symmetrically, when the average CPU usage gets under the minimum threshold, the control loop triggers the removal of one back-end. It contrasts with the static configuration case (without Jade) of a system that is not resized: as the workload increases, the CPU usage saturates rapidly. This results in a trashing of the database, which stops when the load decreases.

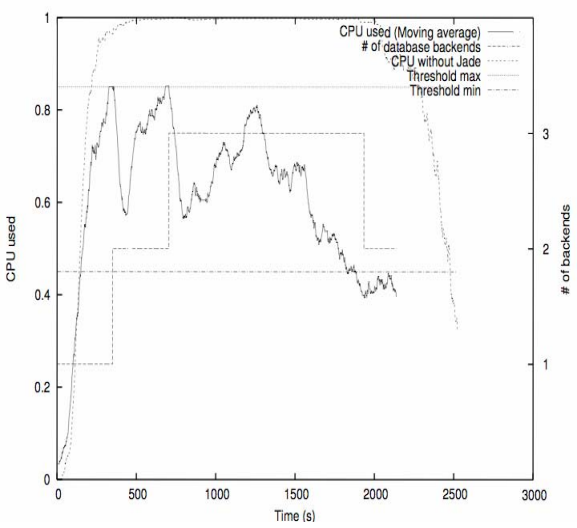


**Figure 3. Behavior of the database tier**

## 4. Related work

Autonomic computing is an appealing approach that aims at simplifying the hard task of system management, thus building self-healing, self-tuning, and self-configuring systems [7].

Management solutions for legacy systems are usually proposed as ad-hoc solutions that are tied to particular legacy system implementations [10, 13]. This unfortunately reduces reusability of management policies and requires these policies to be reimplemented each time a legacy system is taken into account in a particular context. Moreover, the architecture of managed systems is often very complex (e.g. multi-tier architectures), which requires advanced support for its management. Projects such as Jade or Rainbow [6], with a component-based approach, propose a generic way to manage complex system architectures.

Several projects have addressed the issue of self-optimization and resource management in a cluster of machines. Instead of statically allocating resources to applications managed in the cluster (which would lead to a waste of resources), they aim at providing dynamic resource allocation.

In a first category of projects, the software components required by any application are all installed and accessible on any machine in the cluster. Therefore, allocating additional resources to an application can be implemented at the level of the protocol that routes requests to the machines (Neptune [10] and DDSD [14]). Some of them (e.g. Cluster Reserves [3] or Sharc [11]) assume control

over the CPU allocation on each machine, in order to provide strong guarantees on resource allocation.

In a second category of projects, the unit of resource allocation is an individual machine (therefore applications are isolated, from a security point of view). A machine may be dynamically allocated to an application by a hosting center, and the software components of that application must be dynamically deployed on the allocated machine. Projects like Jade, Oceano [2], QuID [9], OnCall [8], Cataclysm [12] or [13] fall into this category.

## 5. Conclusion

This paper has presented the design of Jade, an infrastructure for the autonomic management of legacy distributed applications. The type of applications that Jade addresses are those composed of legacy systems organized within a complex distributed and often replicated architecture.

In this paper we apply this framework to the self-optimization of J2EE applications in the face of the wide load variations observed in Internet applications. More precisely, we use Jade to implement a control loop which adjusts the number of database replicas according to the load on the database tier in a J2EE application.

As soon as legacy software were wrapped in Fractal components, the implementation of control loops was significantly facilitated.

Then, relying on Jade, we showed that dynamic provisioning of nodes, using a simple threshold-based control algorithm, helps regulating the load on the database servers, and thus protects the users again performance degradation due to overload, while avoiding static reservation of resources.

### References


[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, W. Zwaenepoel. Specification and Implementation of Dynamic Web Site Benchmarks. *IEEE 5th Annual Workshop on Workload Characterization*, Austin, USA, 2002.

[2] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar. Oceano: SLA based management of a computing utility. *7th IFIP/IEEE International Symposium on Integrated Network Management*, Seattle, USA, 2001.

[3] M. Aron, P. Druschel, W. Zwaenepoel. Cluster Reserves: a mechanism for resource management in cluster-based network servers. *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 90-101. ACM Press, 2000.

[4] E. Bruneton, T. Coupaye, J.B. Stefani. Recursive and Dynamic Software Composition with Sharing. 7th *International Workshop on Component-Oriented Programming,* 2002, Malaga, Spain. http://fractal.objectweb.org/

[5] E. Cecchet, J. Marguerite, W. Zwaenepoel. C-JDBC: Flexible Database Clustering Middleware. USENIX Annual Technical Conference, Freenix track, Boston, USA, 2004.

[6] D. Garlan, S.W. Cheng, A.C. Huang, B. Schmerl, P. Steenkiste. Rainbow: Architecture-based self adaptation with reusable. *IEEE Computer*, 37(10), 2004.

[7] J. O. Kephart, D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer Magazine*, 36(1), 2003.

[8] J. Norris, K. Coleman, A. Fox, G. Candea. OnCall: Defeating Spikes with a Free-Market Application Cluster. In *1st International Conference on Autonomic Computing*, New York, USA, 2004.

[9] S. Ranjan, J. Rolia, H. Fu, E. Knightly. QoS-Driven Server Migration for Internet Data Centers. In *10th International Workshop on Quality of Service*, Miami Beach, USA, 2002.

[10] K. Shen, H. Tang, T. Yang, L. Chu. Integrated resource management for clusterbased internet services. *5th USENIX Symposium on Operating System Design and Implementation*, Boston, USA, 2002.

[11] B. Urgaonkar, P. Shenoy. Sharc: Managing CPU and network bandwidth in shared clusters. *IEEE Transactions on Parallel and Distributed Systems*, 15(1), 2004.

[12] B. Urgaonkar, P. Shenoy. Cataclysm: Handling Extreme Overloads in Internet Services. Technical report, Dept of Computer Science, University of Massachusetts, USA, 2004.

[13] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal. Dynamic Provisioning of Multi-Tier Internet Applications. *2nd International Conference on Autonomic Computing*, Seattle, USA, 2005.

[14] H. Zhu, H. Ti, Y. Yang. Demand-driven service differentiation in cluster-based network servers. *20th Annual Joint Conference of the IEEE Computer and Communication Societies*, Anchorage, USA, 2001.