Autonomic Energy Management in a Replicated Server System

Aeiman Gadafi, Laurent Broto
Amal Sayah, Daniel Hagimont
Toulouse University
Toulouse, France
Email: LastName@irit.fr

Noel Depalma
Grenoble University
Grenoble, France
Email: Noel.De_Palma@inrialpes.fr

***Abstract*—Nowadays, medium or large-scale distributed infrastructures such as clusters and grids are widely used to host various kinds of applications (e.g. web servers or scientific applications). Resource management is a major challenge for most organizations that run these infrastructures. Many studies show that clusters are not used at their full capacity and that there are therefore a huge source of waste.**

Autonomic management systems have been introduced in order to dynamically adapt software infrastructures according to runtime conditions. They provide support to deploy, configure, monitor, and repair applications in such environments.

In this paper, we report our experiments in using an autonomic management system to provide resource aware management for a clustered application. We consider a standard replicated server infrastructure in which we dynamically adapt the degree of replication in order to ensure a given response time while minimizing energy consumption.

***Keywords*-autonomic computing; administration; energy; replication;**

I. Introduction

Nowadays, medium or large-scale distributed infrastructures such as clusters and grids are widely used to host various kinds of applications (e.g. web servers or scientific applications). The development of *Cloud Computing* [1] illustrates a general trend towards the emergence of large-scale hosting centers.

Power consumption is becoming a major challenge for most organizations that run these infrastructures. According to the U.S. Environmental Protection Agency (EPA) [2], it is estimated that this sector consumed about 61 billion kilowatt-hours (kWh) in 2006 (1.5 percent of total U.S. electricity consumption) for a total electricity cost of about $4.5 billion. Moreover, energy consumption of these infrastructures is estimated to have doubled between 2000 and 2006 and the development of hosting centers will amplify this increase.

In clustered infrastructures, a classical structuring pattern is to replicate servers in order to enforce scalability. In this pattern, a given server is replicated statically at deployment time and a frontend proxy acts as a load-balancer and distributes incoming requests among the replicas. Such a design choices induces resource overbooking to face peak loads while guaranteeing quality of service. As there is a relatively little difference in power consumption between an idle node and a fully used node, the penalty (regarding energy) for keeping an idle node powered on is high. Moreover, such hosting infrastructures require large cooling systems which also consume a lot.

Autonomic management systems [3] have been proposed as one solution for the management of distributed infrastructures. Such systems can be used to deploy and configure applications in a distributed environment. They can also monitor the environment and react to events such as failures or overloads and reconfigure applications accordingly and autonomously.

In this context, we aim at using an autonomic management system in order to dynamically adapt the degree of replication according to the load it receives. This adaptation is a means to dynamically allocate or free machines, i.e. to dynamically turn cluster nodes on - to be able to efficiently handle the load imposed on the system - and off - to save power under lighter load. We implemented such a management policy for a clustered database server in a J2EE application, and evaluated the benefits for energy consumption. Instead of of turning machines on or off which is quite costly, we use an efficient suspension to RAM mechanism to quickly turn the machines in power saving mode.

The rest of paper is structured as follows: Section 2 presents the context of our work and our motivations. Section 3 describes our approach. Section 4 presents the experiments to evaluate our approach. After an overview of related works in Section 5, we conclude in Section 6.

II. Context

In this section, we first present the application case that we use to illustrate our approach. We then overview the autonomic management system that we used in our experiments.

A. Clustered J2EE Application

As experimental environment, we made use of the Java 2 Platform, Enterprise Edition (J2EE), which defines a model for developing web applications [4] in a multi-tiered architecture. Such applications are typically composed of a web

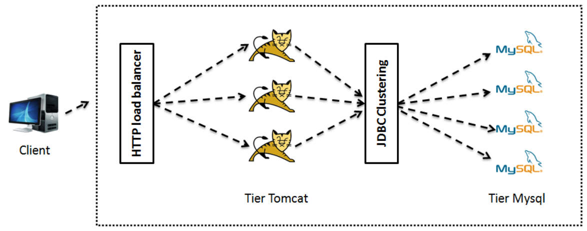

Figure 1. Clustered J2EE servers

server (e.g. Apache), an application server (e.g. Tomcat [5]) and a database server (e.g. Mysql [6]). Upon an HTTP client request, either the request targets a static web document, in this case the web server directly returns that document to the client; or it refers to a dynamically generated document, in that case the web server forwards the request to the application server. When the application server receives a request, it runs one or more software components (e.g. Servlets, EJBs) that query a database through a JDBC driver (Java DataBase Connection driver). Finally, the resulting information is used to generate a web document on-the-fly that is returned to the web client.

In this context, the increasing number of Internet users has led to the need for highly scalable and highly available services. To deal with high loads and provide higher scalability of Internet services, a commonly used approach is the replication of servers in clusters. Such an approach (Figure 1) usually defines a particular software component in front of each set of replicated servers, which dynamically balances the load among the replicas. Here, different load balancing algorithms may be used, e.g. Random, Round-Robin, etc.

In such an architecture, a difficult issue is the find the best degree of replication for each tier, which should be sufficient to tolerate load peaks, but not too high to prevent resource wasting.

B. An Autonomic Management System

A classical pattern in a standard QoS infrastructure is depicted by Figure 2. In such pattern, a given resource R is replicated statically at deployment time and a front-end proxy P acts as a load balancer and distributes incoming requests among the replicas. This kind of architecture induces resource overbooking and therefore energy wasting.

In previous work, we designed and implemented several prototypes of autonomic management systems [7]. These systems are based on an architecture in which sensors are deployed to monitor the environment and autonomic managers which react to these events and perform reconfiguration following autonomics management policies.

In this paper, one important autonomic management policy we target is to minimize energy consumption while meeting the end user needs (i.e. keeping response time

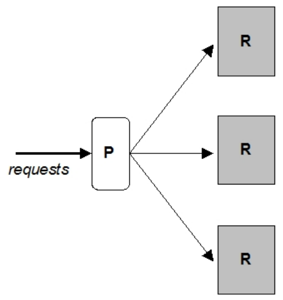

Figure 2. Load balancing among replicas

acceptable). This management system aims at autonomously increasing/decreasing the number of replicated resources used by the application when the load increases/decreases. This has the effect of efficiently maximizing resource utilization (i.e.no resource overbooking) and therefore minimizing the overall energy consumption.

To this purpose, we implemented sensors to probe the CPU usage and the response time of application-level requests and actuators to increase/decrease the number of server replicas.

The autonomic manager makes use of an analysis/decision component based on a minimal and a maximal response time threshold which is responsible for the implementation of the autonomic management policy.

As illustrated in Figure 3, this component receives notifications from sensors and, if a reconfiguration is required, it increases the number of resources by contacting the Cluster Manager to turn cluster nodes on - to handle the load when it raises up - and off - to save power under lighter load.

More precisely, the main operations performed by the resource manager are the following:

- If more resources are required:
  - turn on a new node for the application
  - start a new server replica on the new node
  - Integrate the new replica in the architecture of the application (at the level of the load balancer)
- If resources are under-utilized:
  - Remove one replica from the load balancer
  - Stop this replica
  - Turn off the node hosting this replica

To avoid the system oscillation, an important parameter of the autonomic system is it statibility.

In our case this parameter inhibits any reconfiguration if the system load is too unpredictable. Indeed if the system's load move too quickly around a threshold, turning a machine on or off will be inefficient since the autonomic system will trigger the exactly opposite order just after. In other word, the load of the system must be relatively stable for a certain period of time before reconfiguring the system (i.e. the load must stay stable over/below the system's threshold in order to trigger a reconfiguration).

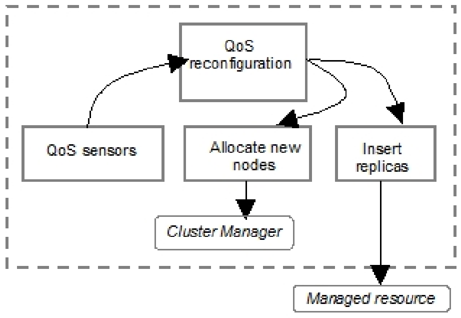

Figure 3. Control loop

III. Autonomic Energy Management

In this section, we present the management policy that we implemented in order to manage energy in a clustered J2EE architecture. This policy applies to the database tier in the J2EE architecture.

A. Management of the replicated database tier

The load-balancer among replicated databases is c-jdbc [8]. The database load, arriving from the web server is distributed by c-jdbc to the DB replicas that can be added and removed based on workload variations. The QoS sensor in this case was at the c-jdbc level. In order to implement the integration of a new database backend in a pool of replicas, we have to consider two case, according to whether the database can be modified or not.

Read-only access In the simplest case, all nodes (on or off) contain the same database content and don't require any special care when integrating a new database server in the database pool.

Read-write access In this more tricky case, we have to take into account the current state of the replicated database. The technique we use leverages the logging facilities of c-jdbc. For each node activation, the manager performs the reconciliation operation on the node, thus bringing it in the same state of all the database replicas. As illustrated in Figure 4, to implement the reconciliation, the log file is used to replay all the SQL statements that have been recorded since the last state synchronization. This is a relatively fast operation given the fact the read/write ratio is high; however it depends on the time between state synchronizations and the number of writes during this time.

B. Suspension to RAM

Turning machines off and (especially) on is quite costly. We measured that such an operation costs about 45 seconds in the average. Instead, we rely on suspension to RAM, which allows to suspend and resume the activity of a machine at a low cost (about 4 seconds in the average for resuming a machine) while saving as much energy as if it was turned off. Suspend-to-RAM stores information on system configuration, open applications, and active files in main memory (RAM), while most of the system's other components are turned off. When a machine is suspended, only the RAM and the network device are power on.

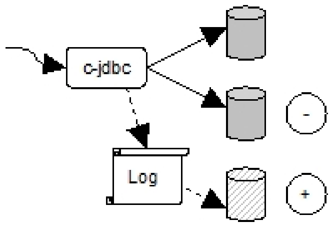

Figure 4. Autonomic replication of database servers

IV. Evaluation

In this section, we present the evaluation of our solution.

Testbed application

The evaluation has been realized with RUBiS [9], a J2EE application benchmark based on servlets, which implements an auction site modeled over eBay. It defines 26 web interactions, such as registering new users, browsing, buying or selling items. RUBiS also provides a benchmarking tool that emulates web client behaviors and generates a tunable workload. This benchmarking tool gathers statistics about the generated workload and the web application behavior.

Hardware environment

The experimental evaluation has been performed on a cluster of x86-compatible machines. The experiments required up to 7 machines: one node for Tune management platform [10] and application server load-balancer, one node for web and servlet servers, one node for database load-balancer, up to three nodes for database servers, one node for RUBiS client emulator (which emulates up to 1000 clients). The number of nodes actually used during these experiments varies, according to the dynamic changes of the workload, and thus to the dynamic resizing of the application. All the nodes are connected through a 100Mbps Ethernet LAN to form a cluster. For energy measurement, we used a programmable power meter (HAMEG HM8115-2).

Software environment

The nodes run version 2.6.30 of the Linux kernel. The J2EE application has been deployed using open source middleware solutions: Jakarta Tomcat 6.0.20 for the web and servlet servers, Mysql 5.1.36 for the database servers, Cjdbc 2.0.2 [8] for the database load-balancer, HAProxy 1.3 [11] for the application server load-balancer. We used RUBiS 1.4.3 as the running J2EE application. These experiments have been realized with Sun's JVM JDK 1.6.0.05. We used the Mysql Connector/J 5.1 JDBC driver to connect the database load-balancer to the database servers.

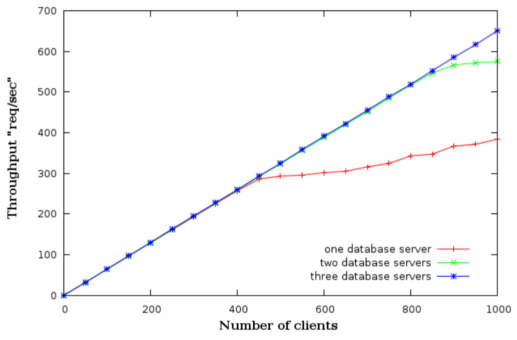

Figure 5. Requests processed per second

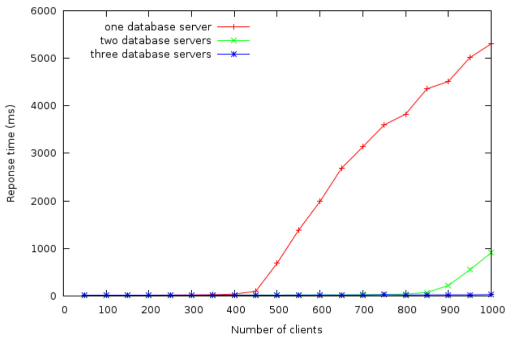

Figure 6. Response time

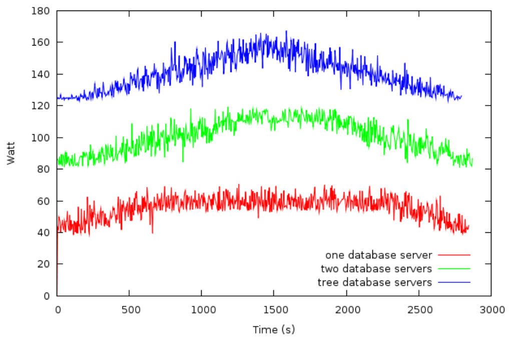

Figure 7. Static configuration

A. Evaluation scenario

We aim at showing that dynamic allocation and deallocation of nodes in response to workload variations allows energy saving with tolerable performance degradation. In this evaluation, we provide measurements for the database replicated tier only.

In our benchmark, the load increases progressively up to 1000 emulated clients: 50 new emulated clients every minute. Since a RUBiS client runs for 25 minutes, the total time of the experiment is about 45 minutes.

During the experiment, the probe monitors the CPU usage of Mysql server nodes. If the CPU usage gets higher than the value given by the *maxCPU* parameter, the probe sends a notification to Tune asking to add a new server. If the CPU usage gets lower than the *minCPU* parameter, the probe sends a notification to Tune asking to remove the Mysql server. The J2EE architecture is initially deployed with one database server (Mysql) and Tune reacts to the load variation by allocating and freeing nodes, as described in section 3.1.

We compare energy consumption and quality of service in two situations:

- static configuration of the J2EE architecture. We measure the performance (regarding quality of service and energy consumption) with one, two and three database servers. Fewer servers will save energy but with a degradation of quality of service. More servers will optimize quality of service, but with energy wasting.
- dynamic configuration of the J2EE architecture. Tune is used to dynamically adapt the number of database servers as described above. Therefore, quality of service is guaranteed without wasting energy.

B. Result

In a first evaluation, we measure the throughput and the response time of the J2EE application when statically configured with one, two and three Mysql servers. These measurements are given in Figure 5 and Figure 6.

In Figure 5, we observe that a single Mysql server can maintain the quality of service for up to 450 clients, and up to 900 clients for two database servers. The configuration with three database servers can maintain the quality of service for all the emulated clients. We observe the same effects in Figure 6 regarding response time.

Figure 7 gives energy consumption of the J2EE application when statically configured with one, two and three Mysql servers. Obviously, the more machines you use, the more you consume.

In conclusion, in this experiment, maintaining the quality of service requires three database servers, but these three servers are not required during the whole experiment, which leads to energy wasting that we evaluate in the following.

With Tune, we dynamically turn cluster nodes on - to be able to handle load peaks - and off - to save power under lighter load. We only use nodes (and consume energy) when needed. We observed that with Tune, throughput and the response time are almost the same as with three database servers, as Tune adapts the number of servers according to the load (more details are given in Figure 9).

Figure 8 shows the consumed energy when using Tune. We start with only one Mysql server, when this server approaches the saturation point, Tune adds a new server to the architecture. This occurs twice at times 600 and 1000.

During this experiment, if we compare the energy consumption of the static configuration with 3 database servers, with the energy consumption with Tune (both being able to

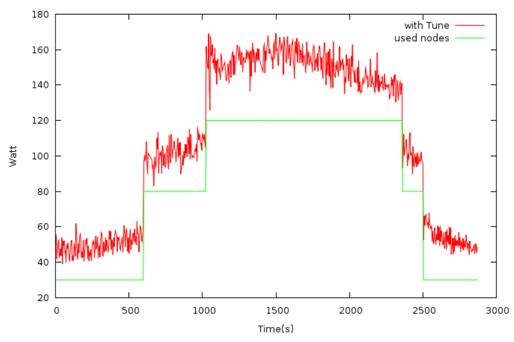

Figure 8. Dynamic configuration with Tune

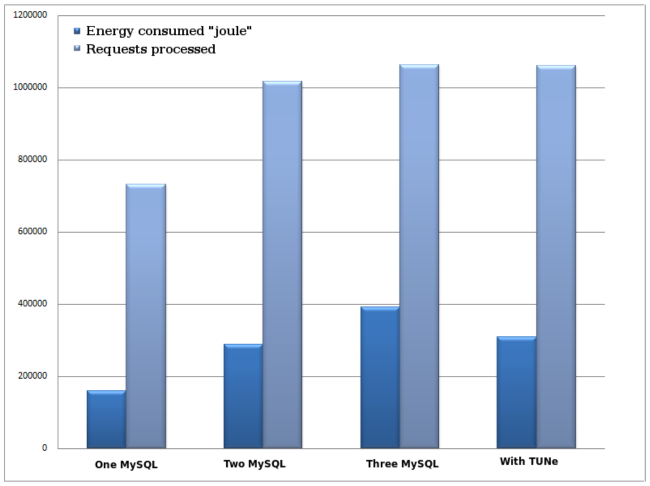

Figure 9. Comparison

maintain quality of service), we observe that Tune reduces energy consumption by 21%.

Another evaluation metric is the number of requests handled by the web server during the experiment. Indeed, when the server saturates, some requests cannot be handled in time and are therefore rejected. Figure 9 gives the number of handled requests during this experiment for the different considered configurations. The left bar gives the consumed energy and the right bar the number of handled requests. We observe that Tune is close to the optimum regarding handled requests and that it provides the best tradeoff between quality of service and energy.

V. Related work

Many works have addressed the issue of power management. Most of them have focussed on energy management for electronic devices powered by electric battery, and few have addressed this issue in grid or cluster infrastructures.

Many research works which aim at managing energy for a single processor system can be used to optimize energy consumption independently on each node of a cluster. This can include optimizations of the use of the processor, memory, disk, etc...

Most of the research that focusses on cluster-wide energy management deals with resource allocation. We can mention some examples of such works:

- **Load balancing** In this category, we can cite the work of E. Pinheiro et al. [12] who developed an algorithm which makes load balancing decisions by considering both the total load imposed on the cluster and the power and performance implications of turning nodes off.
- **Virtualization** In this category, we can cite the work of Hermenier et al. [13] who developed a system which relies on virtual machine migration for transparently relocating any application in the cluster. The placement policy takes into account the cpu and memory usages, in order to concentrate the workload on fewer nodes of the cluster, thus allowing unused nodes to be shutdown. We are currently cooperating with them to integrate our autonomic management system with their work.
- **Simulation** We can here cite the work of Khargharia et al. [14]. They present a theoretical framework and methodology for autonomic power and performance management in data centers. They rely on simulation to apply their approach to two case studies, a multi-chip memory system and a high performance server cluster.

Our work is orthogonal to these contributions. While most of the works made in this domain is specific to energy management, our autonomic computing approach is generic, as it can be used to define any management policy for any distributed software architecture. The field of energy management was not previously addressed by Tune, but the experiment reported in this paper shows that Tune can by used to define an energy management autonomic policy.

The closest work to our is that of Kephart et al. [15] who proposed a multi-agent approach for managing power and performance in server clusters by turning off servers under low-load conditions. Instead of relying on components and architectures, their autonomic system follows a multi-agent paradigm.

VI. Conclusion and Future Work

Nowadays, medium or large-scale distributed infrastructures such as clusters and grids are widely used to host various kinds of applications. Power consumption has become a major challenge for most organizations that run these infrastructures. Many studies show that they are not used at their full capacity and that there are therefore a huge source of wasted power. Autonomic management systems have been recognized as a convenient solution for management of distributed infrastructures.

The experiments that we conducted show that the autonomic computing approach can be used for energy management in a distributed infrastructure. This approach meets the need of energy aware computing, which can be summarized in: minimize power consumption, without affecting the performance of the system. In our experiments, we were able to reduce the power consumption of a typical web application benchmark by approximately 21%.

This paper reported on preliminary work. In the near future, we aim at evaluating much deeply our prototype through more elaborated power management policies, which would include other parameters, for example, network traffic information. We also wish to integrate virtualization techniques in our prototype, as it would enable transparent process (VM) migration between hardware nodes.

Acknowledgments

The work reported in this paper benefited from the support of the French National Research Agency through project SelfXL (ANR-08-SEGI-017-04).

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[2] U. E. P. A. (EPA), "Report to congress on server and data center energy efficiency," August 9, 2007.

[3] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," in *IEEE Computer Magazine, 36(1)*, 2003.

[4] S. Microsystems, "Java 2 platform enterprise edition (j2ee)." [Online]. Available: http://java.sun.com/j2ee/

[5] T. A. S. Foundation, "Tomcat Documentation. The Apache Jakarta Project," http://tomcat.apache.org/tomcat-6.0-doc/.

[6] "MySQL Web Site http://www.mysql.com/."

[7] D. Hagimont, P. Stolf, L. Broto, and N. Depalma, "Component-based autonomic management for legacy software," *Autonomic Computing and Networking*, 2009.

[8] E. Cecchet, J. Marguerite, and W. Zwaenepoel, "C-JDBC: Flexible Database Clustering Middleware," in *USENIX Annual Technical Conference, Freenix track*, Boston, MA, 2004.

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

[10] L. Broto, D. Hagimont, P. Stolf, N. Depalma, and S. Temate, "Autonomic management policy specification in tune," in *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, 2008.

[11] "HAProxy Web Site," http://haproxy.1wt.eu/.

[12] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *In Workshop on Compilers and Operating Systems for Low Power*, 2001.

[13] F. Hermenier, N. Loriant, and J.-M. Menaud, "Power management in grid computing with xen," in *Frontiers of High Performance Computing and Networking ISPA 2006 Workshops*, 2006, pp. 407–416.

[14] B. Khargharia, S. Hariri, and M. S. Yousif, "Autonomic power and performance management for computing systems," *Cluster Computing*, vol. 11, no. 2, pp. 167–181, 2008.

[15] R. Das, J. O. Kephart, C. Lefurgy, G. Tesauro, D. W. Levine, and H. Chan, "Autonomic multi-agent management of power and performance in data centers," in *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 107–114.