A technique for self-optimizing scalable and dependable server clusters under QoS constraints

Ramón Medrano Llamas
IT Department
European Organization for Nuclear Research
Genève, Switzerland
ramon.medrano.llamas@cern.ch

Daniel F. García, Joaquín Entrialgo
Department of Computer Engineering
University of Oviedo
Gijón, Spain
dfgarcia@uniovi.es, joaquin@uniovi.es

***Abstract*—Cluster management has become a multi objective task that involves many disciplines like power optimization, fault tolerance, dependability and online system operation analysis. Efficient and secure operation of these clusters is a key objective of any data center policy. In addition, the service provided by these servers must fulfill a level of quality of service (QoS) to the customers. Applying self-management techniques to these clusters would simplify and automate its operation. Current self-management techniques that take into account service level agreements (SLAs) do not cover at the same time the two most important sides of the cluster operation: self-optimization, for an efficient and profitable operation, and self-healing, for a secure operation and high level of quality of service perceived by users. This work integrates a self-optimization strategy for Internet server clusters that optimizes the power consumption, using dynamic provisioning of servers, with a self-healing strategy that improves the reaction of the cluster to a server failure, by using the spare capacity of the cluster intelligently. The self-management technique is based on empirical response time and power consumption models of the servers that simplify its operation. Additionally, the technique presented in this paper guarantees the fulfillment of the SLA.**

Keywords—self-managing; self-optimizing; self-healing; power optimization; fault tolerance; autonomic computing

I. INTRODUCTION

Nowadays, server clusters are fundamental to support Internet services. Managing large clusters and data centers is a significant problem that involves the fulfillment of multiple objectives, such as providing the service with adequate quality to the customers, reducing the cost of operation of the cluster (which includes, besides the administration and operation budget, the cost of energy and cooling) or reducing the number of catastrophic failures. Applying and developing new techniques that automate the fulfillment of all these objectives is crucial for the effective and profitable operation of these large facilities.

Quality of Service (QoS) is the primary constraint, since it is the main concern for the customers. The QoS limits are agreed by customers and service providers by signing service level agreements (SLA), which include performance metrics and limits to fulfill, functional requirements, penalizations in case of violation, and so on. In this work, we focus on fulfilling the QoS expressed as a limit to the 90-percentile response time, which is a common constrain in SLAs.

There are techniques that focus on maintaining the QoS, without taking into account power consumption and fault-tolerance at the same time. This work introduces a novel technique for self-managing scalable and dependable clusters. The technique is based on the self-optimization of the cluster's power consumption and it applies autonomic management strategies that use simple models of the cluster's servers. The scalability is attained with server provisioning, which modifies the cluster architecture when the workload changes, so that the QoS is maintained. The dependability is achieved by implementing a self-healing strategy that uses the dynamic reconfiguration capabilities of the cluster. This strategy maintains the QoS perceived by the customers of the service when a server fails.

The technique is highly applicable and easy to use, as it is based on simple models of the servers. The experiments performed show that power consumption can be reduced significantly, to values between 17% and 85% of the power consumption of a system that does not use provisioning, depending on the workload. The experiments also show that the QoS is maintained when there is a server failure.

The rest of the paper is organized as follows: section 2 explores the related work. Section 3 describes the self-optimizing strategy, which is closely related to the power management, and the self-healing strategy is presented on section 4. Finally, sections 5 and 6 detail the experimentation and conclusions, respectively.

II. RELATED WORK

There have been numerous approaches to optimize the power consumption of servers and improve their fault tolerance by applying autonomic computing techniques. The initial road map for the development of these techniques was set by [1].

The first works related with the power optimization and automatic management were developed at the level of a single server. These works include utilization threshold management [2] and device optimization through dynamic power configuration of communication links for devices, such as PCI Express, and dynamic frequency scaling (DVFS) of processors [3], [4].

The second generation of self-managing techniques extended the scope of application to the entire cluster. They dynamically provision servers and consider different possible topologies of the cluster that depend on the load that the cluster is supporting [5]. The metrics and models used by the self-optimization manager can vary depending on the objective of the optimization; for instance, in addition to utilization level metrics, metrics based on the temperature can be used to manage topologies that minimize the heat to dissipate and reduce the cost of cooling [6].

The newer generations of power self-managers combine the dynamic provision of servers with the consolidation of virtual machines in a few servers in order to maximize the utilization of the physical servers [7]. They also can manage hierarchically the servers taking decisions over the frequency of the processors in the servers and about the topology of the cluster [8].

Another key component in the self-management strategy is the admission control, since implementing it allows limiting the number of clients in the service; which helps guaranteeing the SLA. Admission control is also improved in the new generation of managers, including predictive and probabilistic methods [9], [10]. However, there are many works in which the implementation of power optimization techniques does not include the warranty of the SLA constraints fulfillment, like on [11] and [12] in which self-management systems for power optimization are introduced, but SLA is not guaranteed.

On the side of the fault tolerance, there are many options commonly used in the industry of distributed software that are custom-made features of the distributed applications. They run as part of the service, like databases or distributed batch computing. Thus, they are not tightly integrated on autonomic managers. A wide review on self-healing techniques is done in [13]. One of the classical approaches is to recover the work done by the worker server that has failed by using temporal checkpoints and re-do logs [14]. However, there are challenges in this approach that must be solved for each new distributed system, like the way to implement the checkpointing and session migration from the faulty server to a new one.

Other options extensively used are based on overprovisioning of resources, like launching more than one worker process to complete one request [15]. The main problem of this is that a replication of grade *n* multiplies by a factor of *n* the number of resources used.

A summary of the related work is presented on the Table I. All the work on self-managing techniques developed until now uses complex models to drive the power manager. These models are difficult to estimate at the beginning of the operation of the manager and do not characterize in an understandable way the performance and response time expected from the servers.

In addition, most of the power management techniques do not guarantee the fulfillment of the service level agreement (SLA) that the provider has with its customers. This warranty is crucial to operate the service properly and implementing power optimization techniques should not introduce SLA violations. As can be seen in the Table I, only two of the techniques analyzed provide SLA warranty.

TABLE I. COMPARISON OF RELATED WORK

	Power man.			**Load bal.**		**Adm.**		**Tolerance**	
	Dynamic provision of servers	**Explicit DVFS management**	**SLA fulfillment warranty**	**Static load balancing**	**Balancing based on QoS constraints**	**Policy-based admission control**	**QoS-based admission control**	**Load migration between servers**	**Others**
[1]	■			■		□			
[3]		■							
[4]	■			■		■			
[5]	■					■			
[6]	■		■	■		■			
[7]	■			■		■			
[8]	■			■					
[9]	■			■		■			
[10]				■		■			
[11]	■		□	■		■			
[12]	■	■		■		■			
[12]				■					■
[14]								■	■
[15]									■
This work	■	□	■		■	■	■	■	

□ Partial support ■ Full support

On the self-optimization topic, there are no techniques that integrate the optimization of power consumption and, at the same time, increase explicitly the ability of the service to recover to failures. In the Table I, the different works analyzed for self-optimization and self-healing are depicted with shadow cells, and common features like load balancing and admission control are also summarized.

Finally, there are other requirements which are desirable for managers like the one described in this paper, for instance, easy integration with legacy and production systems [16].

The technique presented in this paper a) uses directly estimable models that relate the load supported by the server and its expected response time, b) reduces the consumption of the cluster by using the most efficient topology, c) guarantees the fulfillment of the SLA, d) improves the fault tolerance by taking advantage of the spare servers on demand, and d) can be easily integrated with legacy and production software.

III. SELF-OPTIMIZING STRATEGY

Fig. 1 shows the cluster architecture where the technique is applied. The clients' requests are composed of transactions grouped in sessions, whose state is kept in a server; therefore all the transactions of a session have to be handled by the same server. These requests are received by a distributor that acts as load balancer and forwards them to one of the servers in the cluster. The server responses are routed back to the client by the distributor. This distributor is a single point of failure and can become a performance bottleneck if there are many sessions. To address this problem, the distributor can be replicated in several nodes and the sessions can be distributed to each of the distributors using techniques as DNS load balancing.

The technique is implemented in an autonomic manager module integrated in the distributor. In this point of the topology, it can take decisions about the whole cluster, since it will receive all the session requests. To operate, the manager uses models of the response time of each of the servers of the cluster. These models provide the expected response time of a server *i*, R_i, as a function of the number of active sessions on it, N_i. Additionally, the manager uses similar models that provide an estimation of the power consumption of a server *i*, P_i, as a function of the load, N_i. The output of the power model is expressed in watts (W).

In order to get the initial models that will be supplied to the manager an experiment is carried out before beginning the normal operation of the manager. In this experiment, one server of each kind is loaded with an increasing number of sessions. The response time and the power consumed for each number of sessions is recorded. Models for response time and power are fitted to the recorded measurements. With these models, the autonomic manager can estimate the expected response time and power consumption of a server by only using the number of sessions that the distributor has assigned to that server. In this paper we assume that all the sessions are similar, but the approach can be extended to handle different classes of session.

Fig. 2 shows how the response time models can be used to predict the response time of a server under different load situations. In particular, if a server *i* has N_i active sessions, the model can be used to get the expected response time: since the model is a polynomial, a simple evaluation is enough to get the expected response time. As Fig. 2 shows, if the SLA is taken into account, the autonomic manager can use a simple polynomial root finder to know the maximum capacity that the server can support without violating the SLA, M_i. To calculate the relative utilization of the server *i* to the maximum capacity for the SLA, U_i, we simply use the following formula: $U_i = N_i / M_i$.

Using power models is important because efficiency is highly dependent on the load. In general terms, the servers behave more efficiently as the load increases, but they are prone to violate the SLA when operating under high load. Therefore, the self-optimizing strategy must try to maintain the servers at a high load to use them efficiently, but, at the same time, it needs to reserve some free capacity to avoid SLA violations. The strategy uses a threshold $U_{max} \leq 1$ to avoid violating the SLA under high loads.

During the normal operation of a server, it can go through a limited number of states that are shown in Fig. 3.

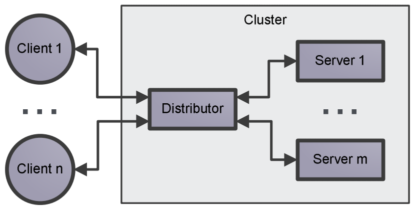


Figure 1. Cluster architecture.

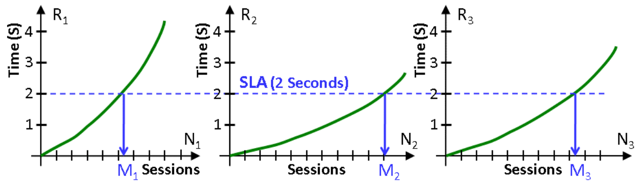


Figure 2. Example of response-time model utilization.

The main operation of the self-optimization strategy takes place when there is some variation in the load supported by the cluster: the opening and closing of sessions. The first event means an increment of the cluster load. If the utilization of all the servers in the cluster is greater than the U_{max} threshold, a new server has to be added. If there are several switched-off servers available, the strategy selects, using the power models, the server that minimizes the expected power consumption of the entire cluster after the server has been activated.

If a new session request is received when all the servers are with a utilization of 1.0 (they are operating at full capacity in relation to the SLA) and there are no more servers available, the load balancer rejects the session. This admission control mechanism allows guaranteeing the SLA, since no server will have more clients than allowed by its response time model.

Fig. 4 shows an overview of all the possible actions of the power manager on the event of an opening of a session. Opening a session involves the admission control, load balancing in case of acceptance, and a power management phase to ensure that the cluster is configured properly to fulfill the SLA while optimizing the power consumption.

The load balancing scheme is based on the utilization of the servers, which is calculated using the response time models experimentally obtained.

The other intervention of the manager is triggered when a session ends and it is shown in Fig. 5. In this case, if the expected utilization after the shutdown of a server is below a threshold called U_{min}, one of the servers that are active (on) should be turned off. In this case, the autonomic manager computes the power consumption of all the possible topologies after a server is turned off, and the most power efficient topology is chosen. Note that this calculation of topologies can yield a lot of combinations on large clusters if it is calculated at server level. An optimization can be performed by calculating at server type level (which is a far more reduced problem in general), since the load on the servers of the same type will be approximately the same.

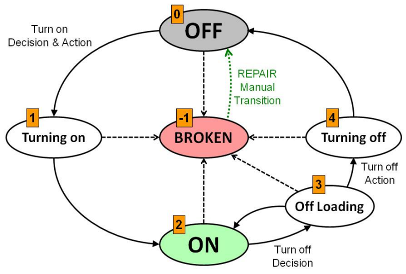
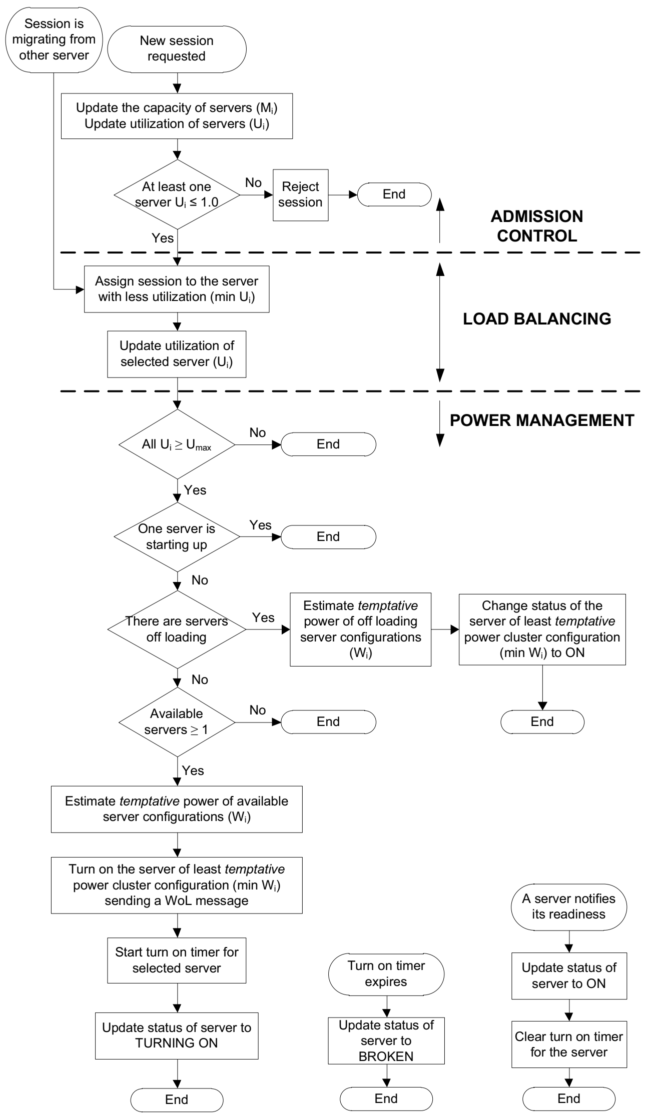


Figure 3. State diagram of the operation of a server.

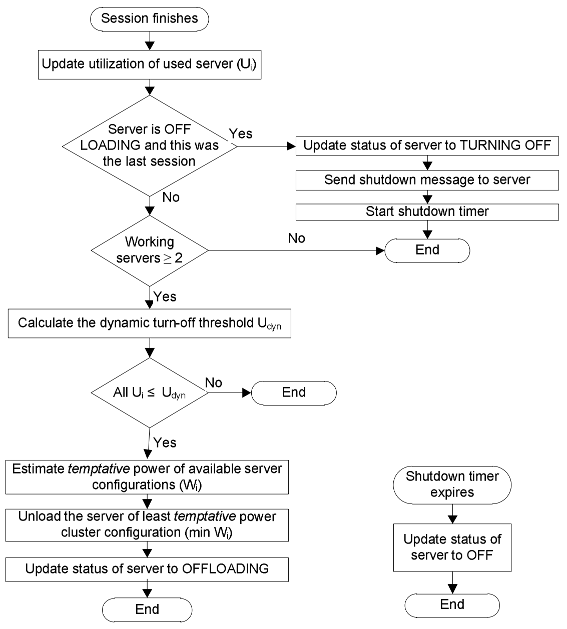


Session is migrating from other server
New session requested
Update the capacity of servers (M_i)
Update utilization of servers (U_i)
At least one server $U_i \le 1.0$
No
Reject session
End
Yes
ADMISSION CONTROL
Assign session to the server with less utilization (min U_i)
Update utilization of selected server (U_i)
LOAD BALANCING
POWER MANAGEMENT
All $U_i \ge U_{max}$
No
End
Yes
One server is starting up
Yes
End
No
There are servers off loading
Yes
Estimate temptative power of off loading server configurations (W_i)
Change status of the server of least temptative power cluster configuration (min W_i) to ON
End
No
Available servers ≥ 1
No
End
Yes
Estimate temptative power of available server configurations (W_i)
Turn on the server of least temptative power cluster configuration (min W_i) sending a WoL message
Start turn on timer for selected server
Update status of server to TURNING ON
End
Turn on timer expires
Update status of server to BROKEN
End
A server notifies its readiness
Update status of server to ON
Clear turn on timer for the server
End

Figure 4. Actions on request and end session events.

The operations for turning off the selected server are depicted on Fig. 5. First the server has to be offloaded, that is, all the sessions assigned to it have to finish before physically turning it off. During this transitory state, no new sessions are assigned to this server. When all the remaining sessions finish, it is turned off with an ACPI command.

One aspect to remark about the offloading of servers is the possibility to change them back to active state when a load peak appears just after a load reduction. Changing a server from offloading to on state is an instantaneous operation. The offloading servers are guaranteed to be the most efficient servers from the set of non-active servers, since when the load was incremented they were turned on because they were the most efficient available.



Session finishes
Update utilization of used server (U_i)
Server is OFF LOADING and this was the last session
Yes
Update status of server to TURNING OFF
Send shutdown message to server
Start shutdown timer
No
Working servers ≥ 2
No
End
Yes
Calculate the dynamic turn-off threshold U_{dyn}
All $U_i \le U_{dyn}$
No
End
Yes
Estimate temptative power of available server configurations (W_i)
Unload the server of least temptative power cluster configuration (min W_i)
Update status of server to OFFLOADING
End
Shutdown timer expires
Update status of server to OFF
End

Figure 5. Operations during a session ending.

IV. SELF-HEALING STRATEGY

The other main objective of the autonomic manager is to minimize the impact of server failures on the response time and availability. To achieve it, the manager takes advantage of the spare capacity that the dynamic provisioning scheme can provide. In case of failure, the manager reduces the reaction time to the failure by substituting the failed server with one of the non-active set, if available.

When a server fails, the first step is to detect the failure as soon as possible to reduce the reaction time to the failure. There are many ways to detect failures, which include, for instance, using heartbeat mechanisms or specialized monitoring software. For the fault tolerance manager proposed here, the detection is done when one of the network procedures fails. Having the cluster working with a very concentrated load by reducing the number of servers that are operating in order to save energy has the side effect that all of the active servers will be serving request with a high throughput; this means that the network requests will have more frequency than the heartbeat messages; then, if a server fails, a network procedure will fail very soon after the server failure.

To detect other types of failures, such as when a server fails when it is turning on, a set of timers have been deployed in different parts of the manager.

After the detection of the failure, all the sessions assigned to the server that has failed are migrated to the servers that are active. This process is done adaptively by migrating each session individually through the load balancer module. The difference in the balancing between a regular session request and a failed session is that, since the session is already accepted, it will never be rejected. Thus the admission control is disabled for balancing the failed sessions.

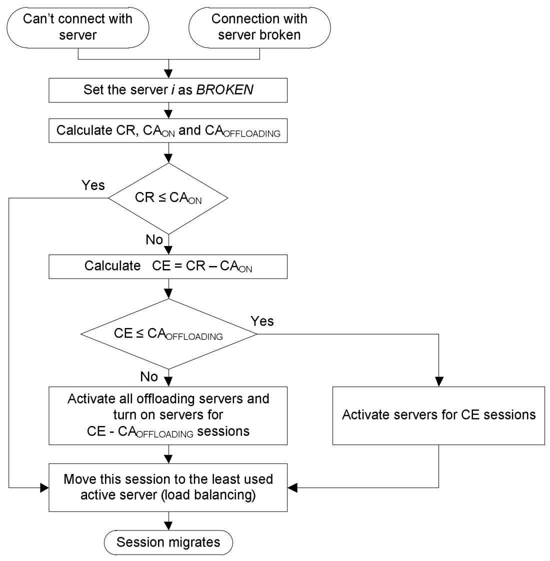

Can't connect with server
Connection with server broken
Set the server *i* as *BROKEN*
Calculate CR, CA$_{ON}$ and CA$_{OFFLOADING}$
CR ≤ CA$_{ON}$
Yes
No
Calculate CE = CR – CA$_{ON}$
CE ≤ CA$_{OFFLOADING}$
Yes
No
Activate all offloading servers and turn on servers for CE - CA$_{OFFLOADING}$ sessions
Activate servers for CE sessions
Move this session to the least used active server (load balancing)
Session migrates

Figure 6. Operations for the topology update on session migration.

During the migration of failed sessions, a reconfiguration of the cluster topology may be needed in order to have enough capacity to fulfill the SLA restrictions after the failure of a server. Fig. 6 summarizes how to perform the topology reconfiguration. The manager will calculate the Capacity Required (*CR*) as the number of sessions assigned to the server that has failed. It also calculates the Capacity Available (CA_{ON} and $CA_{OFFLOADING}$) as the free capacity in the servers that are currently active (on) and offloading.

CA_{ON} is the number of failed sessions that the cluster can serve without any change in the topology while fulfilling the SLA. If *CR* is greater than CA_{ON}, the manager will need to activate more servers to get enough capacity to recover from the failure and serve the sessions already admitted.

When turning on a server, there is a delay from the decision to the final availability of the server. Thus, when a recovery needs to turn a server on, there is a time interval, while waiting for the new server to be available, where there are more active sessions than the SLA permits. This transitory period is the only time in which the SLA could be violated (all the migrating sessions are assigned immediately to active servers to avoid an increase in the response time).

V. Experiments

The experimentation has been designed with two goals: validating how the power self-optimizing strategy scales the cluster and testing the self-healing strategy under different scenarios of operation. Table II details the main experimental conditions. All the experiments are carried out using a synthetic server software. For the sake of brevity, only the results of two experiments will be presented: one related to self-optimization and another to self-healing.

Fig. 7 shows a summary of the results for a power self-optimizing experiment centered on the scalability provided by the autonomic manager. The experiment selected was based on a load ramp up followed by a ramp down that allows the servers to pass through all the states of operation.

TABLE II. Experiment conditions for the validation

Parameter	Value
Servers on at start	1 (high capacity)
Total servers	4 (1 very high, 1 high and 2 normal capacity)
SLA	2 s for the 90-percentile of response time
Max. capacity	378 sessions
Max. load requested	400 sessions
Duration	40,000 s
U_{max}	0,9
U_{min}	0,85

The cluster started with all its servers off, except for one of them, which was not the most efficient. This configuration was chosen in order to let the manager select the most efficient configuration, as can be seen at the end of the experiment.

This experiment was run on a cluster with heterogeneous configuration. The servers were of three different types: 1) very high capacity, using a 2009 Intel Xeon processor with a disk cabinet to improve the I/O performance and providing 180 concurrent sessions, according to the SLA limit; 2) high capacity, using the same processor as before, but with an internal disk, which, due to the less powerful I/O system allowed up to 100 sessions; and 3) normal capacity, using a 2006 AMD Opteron processor with internal disk and providing support for 49 sessions.

The results depicted in Fig. 7 show that the SLA is fulfilled by having some of the sessions rejected when the load requested is over the capacity limit set by the SLA. They also show that the power consumption of the cluster is notably reduced, since the servers are turned on and off as needed. The power savings in this case where of 42.25%, but can go between 17% and 85% in different workloads.

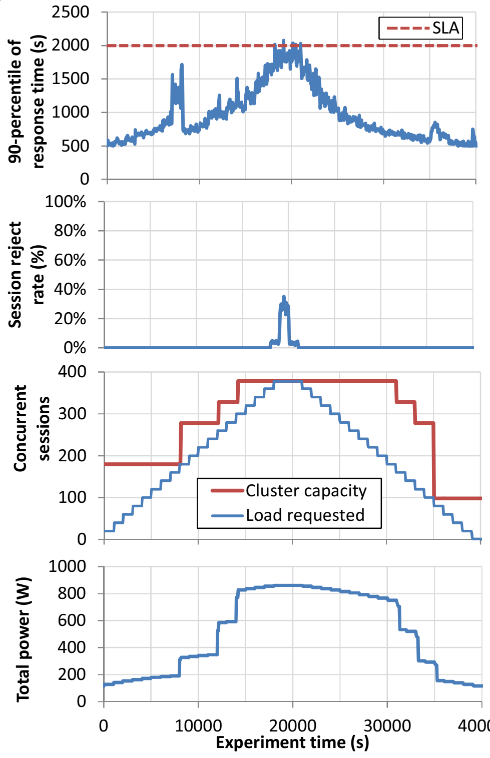


Figure 7. Results of the power management experimentation.

Regarding the self-healing strategy validation, the experiments were centered on studying how the designed strategy performs on different fault events, depending on the configuration of the cluster and the load requested at that moment. Table III summarizes the experiments carried out.

The most interesting aspect of the self-healing strategy is the transitory period in which the SLA could be violated, which has to be minimized in order to maintain the QoS.

Fig. 8 shows the detailed view of the cluster capacity and the active sessions just after a server failure. In this experiment (corresponding to the third row in Table III), the failure happened when the cluster was being used at its full capacity, so no server was in offloading state or available in off state. This situation is the worst for the duration of the transitory period. The results show that the transitory period under these conditions is half of the mean session time.

As Fig. 8 shows, when the server fails, the capacity of the cluster is reduced by the capacity provided by this server, but the number of active sessions is not changed, since they cannot be rejected after they have been accepted. As the users close their sessions, the load supported by the cluster goes down. Note that, during this period, since all the servers are working above full utilization, no new sessions will be accepted (more sessions would make the situation worse) until the load is under 1.0. The violation of the SLA during this period is possible, since the servers are working with more load than they support to fulfill the SLA.

In order to measure the impact of a failure on the SLA, a QoS tracker was added to the manager. This tracker collected statistics by using a sliding window mechanism of the latest transactions processed. For summarizing the SLA violations, the metric selected was the number of windows in which the SLA was violated per user. This metric showed that only in 0,442% of windows there were SLA violations.

VI. Conclusions

This paper has presented a self-optimizing technique for managing large Internet server clusters by including an autonomic manager that manages the cluster topology to optimize power and reaction time to server failures. The strategy to optimize the power consumption guarantees, at the same time, the fulfillment of the SLA restrictions. Also, using the spare capacity wisely it can reduce the reaction time to recover from the failure of a server, improving the QoS provided to the customers. The technique is based on simple models of response time and power. Thus, it is highly applicable and easy to use.

In the future work there are improvements, such as the management of more granular power states and the use of dynamically adapted models for the response time and the power of the servers of the cluster.

TABLE III. Test cases to validate the self-healing strategy

Load state	Load type	Feature to test
Medium load	Static	Use of available capacity
High load	Static	Turning on of servers
Overload	Static	Admission control
Ascending load	Dynamic	Capacity adaptation
Descending load	Dynamic	Recovery of offloading servers

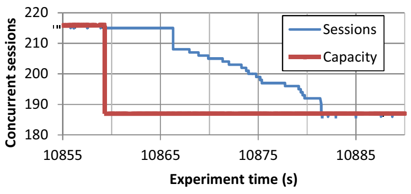


Figure 8. Detail of an adaption time during a failure.

Acknowledgment

This research has been partially supported by the projects TIN2008-06045-C02-01 and TIN2011 24903 of the Spanish National Plan of Research and Development.

References

[1] Pinheiro, E.; Bianchini, R.; Carrera, E.V.; Heath, T.; “Load balancing and unbalancing for power and performance in cluster-based systems”, Workshop on Compilers and Operating Systems for Low Power, vol. 180, pp. 182-195, 2001.

[2] Yung, H.L.; Chung, E.; Simunic, E.; Benini, L.; De Micheli, G.; “Quantitative Comparison of Power Management Algorithms”, Stanford University, 2000.

[3] Kim, K.H.; Beloglazov, A.; Buyya, R.; “Power-aware Provisioning of Cloud Resources for Real-time Services”, MGC ’09, 2009.

[4] Bertini, L., Leite, JCB and Mossé, D.; “Statistical QoS guarantee and energy-efficiency in web server clusters”, 19th Euromicro Conference on Real-Time Systems, 2007. ECRTS'07. 83—92 (2007)

[5] Rusu, C.; Ferreira, A.; Scordino, C.; Watson, A.; , "Energy-Efficient Real-Time Heterogeneous Server Clusters," Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE, vol., no., pp. 418- 428, 04-07 April 2006.

[6] Ramos, L.; Bianchini, R.; “C-Oracle: Predictive Thermal Management for Data Centers”, Rutgers University, 2008.

[7] Wang, Y.; Wang, X.; Chen, M.; “PARTIC: Power-Aware Response Time Control for Virtualized Web Servers”, IEEE Transactions on Parallel and Distributed Systems, 22(2), 2011.

[8] Wang, X.; Wang, Y.; “Coordinating Power Control and Performance Management for Virtualized Server Clusters”, IEEE Transactions on Parallel and Distributed Systems, 22(2), 2011.

[9] Aweya, J., Ouellette, M., Montuno, D.Y., Doray, B. and Felske, K.; “An adaptive load balancing scheme for web servers”, International Journal of Network Management, 12. 3—39 (2002)

[10] Cherkasova, L.; Phaal, P.; “Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites”, IEEE Transactions on Computers, 51(6). pp. 699-685, 2002.

[11] Bertini, L., Leite, JCB and Mossé, D.; “Statistical QoS guarantee and energy-efficiency in web server clusters”, 19th Euromicro Conference on Real-Time Systems, 2007. ECRTS'07. 83—92 (2007)

[12] Chen, J., Huan, K. and Thiele, K.; “Power Management Schemes for Heterogeneous Clusters under Quality of Service Requirements”, ACM SAC ’11 Taiwan

[13] Psaier, H. and Dustdar, S.; “A survey on self-healing systems: approaches and systems”, Computing, vol. 91, pp. 43-73, 2011.

[14] Ghemawat. S.; Gobioff, H.; Leung. ST.; “The Google file system”, In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03), 2003.

[15] Dean, J.; Ghemawat, S.; “MapReduce: simplified data processing on large clusters”, Commun. ACM 51, 1 (January 2008), 107-113, 2008.

[16] Kalbarczyk, Z.T., Iyer, R.K., Bagchi, S. and Whisnant, K.; “Chameleon: A Software Infrastructure for Adaptive Fault Tolerance”, IEEE Transactions on Parallel and Distributed Systems, 10(6). 560—579 (1999)