

Load-Balancing Metric for Service Dependability in Large Scale Distributed Environments (Extended Abstract)

Florin Pop, Marius-Viorel Grigoras, Ciprian Dobre, Ovidiu Achim, Valentin Cristea

Faculty of Automatic Control and Computers, Computer Science Department,
University “Politehnica” of Bucharest, Romania

Emails: florin.pop@cs.pub.ro, ciprian.dobre@cs.pub.ro, ovidiu.achim@oracle.com, valentin.cristea@cs.pub.ro

Abstract—In this paper we present an effective solution to satisfy service dependability in large scale distributed environments. The solution is based on service encapsulation in a special container and fault masking, if appear. The aim is to increase the availability of services and, in the same time, to ensure better use of distributed resources. We highlight in this paper the need for increasing dependability by enhancing state of the art in the field. We describe the service-based architecture and we present the load balancing policy used for replicated services. The performance tests made in real scenarios show the performance of proposed solution. The novelty of proposed solution is represented by new metric for multi-criteria load balancing in a fault tolerant distributed environment based on replication.

Keywords

Services, Dependability, Load-Balancing, Service Reputation, Distributed Systems

I. INTRODUCTION

The concept of dependability [1] for large scale distributed systems is very complex and has been refined over many years of research. It comprises concepts as availability, reliability, safety, confidentiality, integrity and maintainability. *Availability* is the property of a system to offer correct service on demand. *Reliability* is the property of a system to offer correct service continuously. *Safety* is the absence of catastrophic effects on the environment, even in case of incorrect service. *Confidentiality* is the absence of unauthorized disclosure of information. *Integrity* is the absence of improper system state alteration. *Maintainability* is the property of a system to be easily repaired and modified. *Security* is the concurrent existence of availability to authorized users only, confidentiality, and integrity.

In this paper we present a solution to satisfy the service dependability in large distributed environments. We propose a balancing metric to increase availability and a reputation model in order to satisfy QoS. Proposed architecture realizes and optimization access to distributed services and its replicas without to influence scalability applications used in distributed environments. The solution stands out by the support given to services providers, by implementing a system that can be implemented within services container.

The paper will be structured as follows. Related works are considered to have the same goal as this paper are described in Section 2. In Section 3 is considered service based architecture and are highlighted why it was chosen to implement Axis technology solution and architectural solution chosen is explained. Implementation details are explained in detail in Section 4. We talk about load balancing metric used for replicated services in Section 5. In Section 6 are presented performance tests, real scenarios of use and performance and results. The paper ends with conclusions and some possible future works.

II. RELATED WORK

In the Computer Science scientific literature was studied and proposed different solutions based mainly on the change implementation services to ensure fault tolerance.

The study realized by Zhang et al. in [8] proposes to use primary-backup mechanism to ensure fault tolerance and maintaining secure service replicas [9]. Primarybackup is a well-known technique for making services highly available [2], [6], [7]. A client sends a request to the primary, which receives and executes the request. The primary then sends a state update message to the backups and replies to the client. Typically, the primary does not reply to the client until it knows that all backups have received the state update. This is done to ensure that the backups are always consistent with the client: it is impossible for the client to know that the primary executed the request without the backups also knowing this.

There are some approaches that fault tolerance assuring is achieved at TCP [5]. This approach is the starting point in trying to achieve a system-level fault tolerance application. The difference is that in context of services, at application level exists and a context of the message, that contains meta-information about the request, very useful information for fault-tolerance mechanism.

III. SYSTEM ARCHITECTURE

The main improvement that our solution brings is a new policy for load balancing and fault tolerance. We have changed the load balancing policy wanting to look at all parameters of services container. For example, Monitor Service will respond

with a “better” replica because will be capable to take real time decisions based on specific information from each container of a replica.

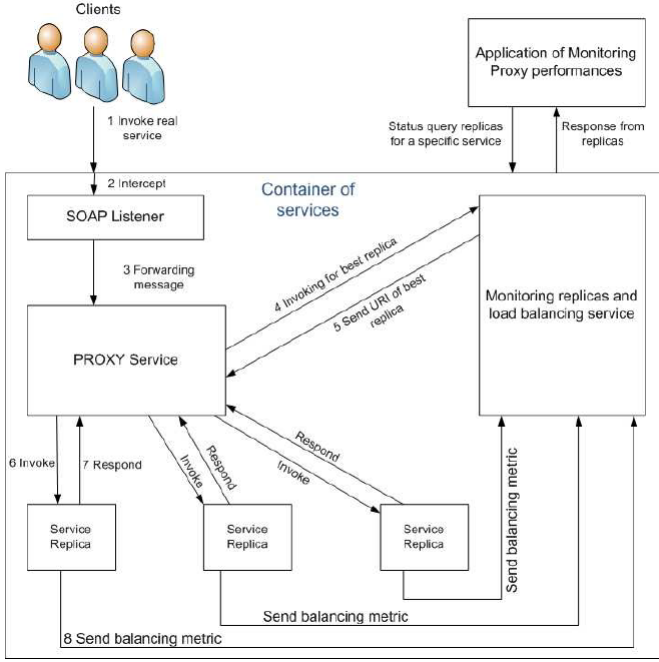


Fig. 1. System Architecture

The proposed architecture implements proxyPattern design pattern [3]. The main objective of this pattern is to define a proxy object, what is positioned between the client and the real service for controlling the access at the real service and for realizing certain processing each time when a service is accessed (see Figure 1). Services container functions like a proxy for replicated services. Messages for real services will be intercepted by container (with a SOAP listener) and will be forwarded to a replica of a real service via through a Proxy Service. Actions realized by container will be transparent both client and service. The client will consider in all time of communication that it is connected at the real service, not at the proxy service, but replicas will respond at requests like they will came from clients side, not for another service side.

IV. LOAD BALANCING METRIC

To achieve the goal of this project, making reliable services, we implemented in each container and a method for calculate a balancing metric which help us assuring load balancing and fault tolerance. Load balancing functionality of replicas optimizes resources utilization and minimizes the response time of a request.

A. Balancing Metric

Each replica of a service has a balancing level calculated according to information gathered by website monitoring tool. The balancing Metric will be a value between 0 and 100. The greater is this parameter, the higher is the hardware performance of the replica container overall. Balancing metric is calculated using following formula:

$$BalancingMetric = \sum_{i=1}^6 P_i * B_i$$

In this formula we consider some criteria for load-balancing, each criterion having a weight in the *BalancingMetric* formula.

B_1 represents the measure of system usage. It consider the number of threads in the system. *Live threads* represent number of threads which are live (it run on the processor or fallows to be scheduled by the operating system waiting at a blocking condition); the term where this parameter is located is calculated versus the total threads supported by that container (default in apache tomcat total threads is equal with 200). The B_1 term is:

$$B_1 = 1 - \frac{LiveThreads}{TotalThreads}$$

B_2 represents a measure of system quality. *Number of errors* represents total number of errors reported by services container. It grows when a request for a replica in that container is finished with error; the term where this parameter is located is calculated versus the total number of requests. *Number of requests* represents total number of requests and help to calculate the second term. We increment this number when receive a new request. The B_2 term is:

$$B_2 = 1 - \frac{ErrorsNo + 1}{RequestsNo + 1}$$

B_3 show the how the processors are occupys in the system, considering the JVM. *Number of processors* represents total number of processors exited in system of the services container; the term that contained this parameter is calculated dividing this number at max number of processors, that is a generic number, it represents the max number of processors that a system can have (we chose this number 8, so a system can be maxim an opteron system); this parameter tells us how much we can parallel services from container a better throughput. *JVM CPU Usage* represents as percent how much JVM usage occupys from all processors. Max value for this parameter is $ProcsNo * 100\%$; when this sixth term tends to 1 means that all processors are almost in sleep mode, so JVM has no process in running mode and when tends to 0.5 means that from all processors, only one is half free, mean that OS can schedule and other processes and the throughput has still an acceptable value. The B_3 term is:

$$B_3 = 1 - \frac{JVMCPUUsage(\%)}{100 * ProcsNo}$$

B_4 represents a way to measure how total free space we have from total memory. When this report will tend to 0, GC (Garbage Collector) will try to delete all unreferenced objects or memory scheduler will try to allocate extra memory for total memory or max memory. Both operations have a high overhead. *Free memory* represents total number of mega bytes free from JVM. *Total Memory* represents total number of mega bytes allocated from Max Memory.

So, the B_4 term is:

$$B_4 = \frac{FreeMemory}{TotalMemory}$$

B_5 represents the proportion of CPU utilization. The B_5 term is:

$$B_5 = \frac{ProcsNo}{MaxProcsNo}$$

B_6 represents the measure of overhead introduced by swap memory. *Free Swap* represents total number of free mega bytes located on the swap partition. Is important to consider and this parameter because swap-in and swap-out operations have a high overhead also. The B_6 term is:

$$B_6 = \frac{FreeSwap(\%)}{100}$$

B. Reputation function for Service Replica

Reputation-based function addresses one of the most critical issue in Grid Environment: QoS for grid-based services. Simulation and, in the future, implementation of Reputation solution over Proxy in Service Environments, will constantly follow to ensure QoS in term of execution times and performance. Solution will permanently monitor the “quality” of the grid and how it’s responding to consumers’ requests and will propose ways to choose either for “best (shortest) execution time” or for “most determinist execution time”. In the first approach, solution will try, based on execution history, to provide with the best service endpoint in term of execution performance. Second scenario will cover those consumers that care more about a deterministic execution time rather than the shortest execution time. Allocation of Services is limited due to little information available for the Monitor. The replica selection algorithm doesn’t know whether a particular service is running a job or if has or not multiple jobs queued. Moreover, it doesn’t have a history of the completion times of the jobs. This makes difficult the prediction of the performance of Web Services. Allocation of services in grid can be improved with a help of a reputation system.

V. TEST SCENARIOS AND EXPERIMENTAL RESULTS

We have realized experimental tests to highlight each term behavior of the proposed metric for balancing. The test conditions consider Lambda probe [4] integrated module write in log files every 30 seconds, balancing metric calculated every approximate 30 seconds (30 s some sync times), an event takes place every 15 seconds (an event can be a request/error, event of GC or of memory scheduler, increasing/decreasing free memory etc). In all test we have used the same Tomcat container that mean that we consider the same resources. The experiments consider the following scenarios:

- Empty container
- Threads Alive
- Successful Requests
- Error requests

- Best Replica
- Many Requests

VI. CONCLUSIONS

The work presented in this paper is concerned by increasing reliability and availability, particularly in Grids and Web-based distributed systems. The presented solution presented is based on encapsulation of replicated distributed services in a container for masking the possible defects that may occur. Thus, for a client, a failure occurred is imperceptible, presented service being designed for masking possible errors and in transparent mode to redirect any received requests to another functional service/replica. Our solution uses a load balancing system that ensures the use of resources more efficiently and reduces the time response.

Using a service by clients assume to know only the service address implemented in the same container with proxy service. This address remains unchanged even if other replicas of the service join in the system or leave from the system. Using transparent replication makes the system more scalable.

ACKNOWLEDGMENTS

The research presented in this paper is supported by national project: ”SORMSYS - Resource Management Optimization in Self-Organizing Large Scale Distributes Systems”, Project CNCISIS-PN-II-RU-PD ID: 201 (Contract No. 5/28.07.2010). The research is co-founded by national project ”DEPSYS - Models and Techniques for ensuring reliability, safety, availability and security of Large Scale Distributes Systems”, Project ”CNCISIS-IDEI” ID: 1710 (618/15.01.2009).

REFERENCES

- [1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Vytautas. Fundamental Concepts of Dependability, 2000.
- [2] K. P. Birman, T. A. Joseph, T. Raeuchle, and A. El Abbadi. Implementing fault-tolerant distributed objects. *IEEE Trans. Softw. Eng.*, 11:502–508, June 1985.
- [3] Huawen Li and Qingjie Wang. Proxy pattern informatization research based on saas. In *Proceedings of the 2009 IEEE International Conference on e-Business Engineering, ICEBE '09*, pages 518–521, Washington, DC, USA, 2009. IEEE Computer Society.
- [4] X. Lu, Q. Yue, Y. Zou, and X. Wang. An Experimental Analysis for Memory Usage of GOS Core. In *Parallel and Distributed Computing, Applications and Technologies, 2008. PDCAT 2008. Ninth International Conference on*, pages 33–36. IEEE, 2008.
- [5] Manish Marwah, Shivakant Mishra, and Christof Fetzer. Fault-tolerant and scalable tcp splice and web server architecture. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, pages 301–310, Washington, DC, USA, 2006. IEEE Computer Society.
- [6] Daniel J. Scales, Mike Nelson, and Ganesh Venkitachalam. The design of a practical system for fault-tolerant virtual machines. *SIGOPS Oper. Syst. Rev.*, 44:30–39, December 2010.
- [7] Dmitrii Zagorodnov, Keith Marzullo, Lorenzo Alvisi, and Thomas C. Bressoud. Practical and low-overhead masking of failures of tcp-based servers. *ACM Trans. Comput. Syst.*, 27:4:1–4:39, May 2009.
- [8] X. Zhang, D. Zagorodnov, M. Hiltunen, K. Marzullo, and R. Schlichting. Fault-tolerant grid services using primarybackup: Feasibility and performance, 2004.
- [9] Qian Zhu and Gagan Agrawal. Supporting fault-tolerance for time-critical events in distributed environments. *Sci. Program.*, 18:51–76, January 2010.