LOAD-BALANCING METRIC FOR SERVICE DEPENDABILITY IN LARGE SCALE DISTRIBUTED ENVIRONMENTS

FLORIN POP*, MARIUS-VIOREL GRIGORAS†, CIPRIAN DOBRE‡, OVIDIU ACHIM§, AND VALENTIN CRISTEA¶

Abstract. In this paper we present a load-balancing metric proposed to increase the availability of services in large scale distributed environments. The services are encapsulation in a special container in order for mask the possible fault. Service availability is one of the important characteristic of service dependability. The proposed metric ensures a better use of resources in distributed systems. We highlight in this paper the need for increasing availability by enhancing state of the art in the field. We describe the service-based architecture that uses a proxy and services replicas and we present the load-balancing metric used by proxy in order to select a service replica. The performance tests were made in real scenarios and show the performance of proposed solution considering some scenarios that inject some faults in the system. The novelty of proposed solution is represented by a new metric for multi-criteria load-balancing in a fault tolerant distributed environment based on replication.

Key words. Web Services, Dependability, Load-Balancing, Service Replication, Distributed Systems.

AMS subject classifications. 68M14, 68M15, 68Q10

1. Introduction. Distributed systems play an important role as a background support in many areas such as trade, communication, science, government and even entertainment. Web service architecture approach in large distributed environments was preferred for interaction with users because this approach presents many advantages such as ergonomic behavior, portability, availability and most import access from different platforms. All elements regarding Web Services design is important in distributed systems [6]: standard structure of messages sent between clients and services, description method, discover method, invoke method and development.

With the increasing interest for distributed systems, also customers' requirements become complex and address a SLA. For example, users are interested for invoking a specific service which answers correctly at their requests in a reasonable time with a specific degree of accuracy. Also, time availability of service is very important: as the service is available to customers over time, the service will be more used. Services that fill these requirements are in dependable services category [13].

The concept of dependability [1] for large scale distributed systems (LSDS) is very complex and has been refined over many years of research. Dependability addresses 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

*Computer Science Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, Email: florin.pop@cs.pub.ro

†Computer Science Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, Email: viorel.grigoras@cti.pub.ro

‡Computer Science Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, Email: ciprian.dobre@cs.pub.ro

§Oracle TSBU, Bucharest, Romania, Email: ovidiu.achim@oracle.com

¶Computer Science Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, Romania, Email: valentin.cristea@cs.pub.ro

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. Availability and reliability are essential in distributed systems, especially systems considered at large scale because customers expect system be operational despite technical problems and work as its specification even though some components were damaged.

In this context, the QoS could be considered as a quantitative measure of service dependability. There are many ways to increase the QoS [13], like: bug fixing before deployment, mechanisms for fault tolerance, mechanisms for recovering from error, etc. In the development process, most developers choose the solution less expensive and faster in detriment of quality for service device. By contrast, service providers want to offer reliable service to customers, to ensure a certain level of QoS. Although the importance of achieving safe services in distributed environments is recognized and many research projects try to find a solution for this problem.

In this paper we present a solution to satisfy the service dependability in LSDS. We propose a balancing metric to increase availability and a reputation model in order to satisfy QoS. The proposed architecture realizes an optimized access to services and its replicas without to influence scalability. The solution stands out by the support given to services providers, by implementing a system as a service container.

The paper is structured as follows. Related works are described in Section 2. Section 3 considers the service based architecture and highlights the technology solution chosen in order to implement the proposed architecture. Implementation details are explained in Section 4. We present in Section 5 the load-balancing metric used for replicated services. In Section 6 are presented the performance tests, scenarios used and performance results. The paper ends with conclusions and some possible future works.

2. Related Work. The study realized by Zhang et al. in [17] proposes to use primary-backup mechanism to ensure fault tolerance and maintaining secure service replicas [18]. Primarybackup is a well-known technique for making services highly available [3, 12, 16]. In this context, 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 [10]. 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.

In [11] is present a system that ensure services dependability what implies that a group of network nodes will be appear to clients as single node. At receiving a request from a client node system choose the nearest node for handling the request. Node choice is performed using a amended version of anycast routing scheme by using properties Mobile IPv6 protocol provided by [14]. Although is a solution that solves many of dependable systems problems, it works only nodes that are XtreemOS system operated [11]. In terms of client system its operation must support Mobile IPv6 protocol.

The solution proposed in [7] is based on developing a system by alternative services implemented differently. Each available service is designed to face a different set of errors, thereby increasing cases where a request can be answered from one of the services available. This solution is based DESL, language for express specifications of each service separately. Specifications for this language are not yet finished.

Another project which proposes load-balancing and fault tolerance is [2]. Here is a comparison between Checkpoint-recovery and balancing using Intra-cluster load-balancing, Intra-grid load-balancing solutions. This paper described a fault detector that detects the occurrence of resource failures and a fault manager that guarantees that the tasks submitted are completely executed using the available resources. Model used in implementation of this fault tolerance technique is existing Intra-cluster and Intra-grid load-balancing model [15].

A project that aims to define and to implement a framework at application level for development of grid dependable services is DIGS [4, 5]. In this project is described the structure of a *Proxy* that must combining several identical services from functionality view in another "better" service. A better service means a service with a higher level of dependability. The starting point objective was to design a mechanism to intercept transparent SOAP messages exchanged between client and server. A client sends messages to proxy like this would be a real service. Proxy intercepts the message and processes it based on a fault tolerance model and it then sends to a real service. Messages from the server to the client are transmitted also via proxy, which can process the response, if any, depending on the fault tolerance implemented model.

3. System Architecture. The main improvement that our solution brings is a new metric for load-balancing and fault tolerance. We have proposed the load-balancing metric in order to look at all parameters of services container. The proposed system architecture is presented in Figure 3.1. For example, the 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.

The proposed architecture implements *ProxyPattern* design pattern [8]. 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. 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.

SOAP Listener is a system component responsible with SOAP messages monitoring received by services container. This component monitories SOAP messages and verifies if these messages must be redirecting to another proxy service or the real invoked service. Situations when a message is transmitted to the real invoked service are:

- Messages that have forwarded by proxy to a replica of invoked service that is implemented in the same container with Proxy, must not be sent to the proxy anymore, to exclude an infinite loop. These messages are marked by Proxy with a specific attribute in the SOAP message header.
- SOAP Messages that have as target some specific special services like Admin

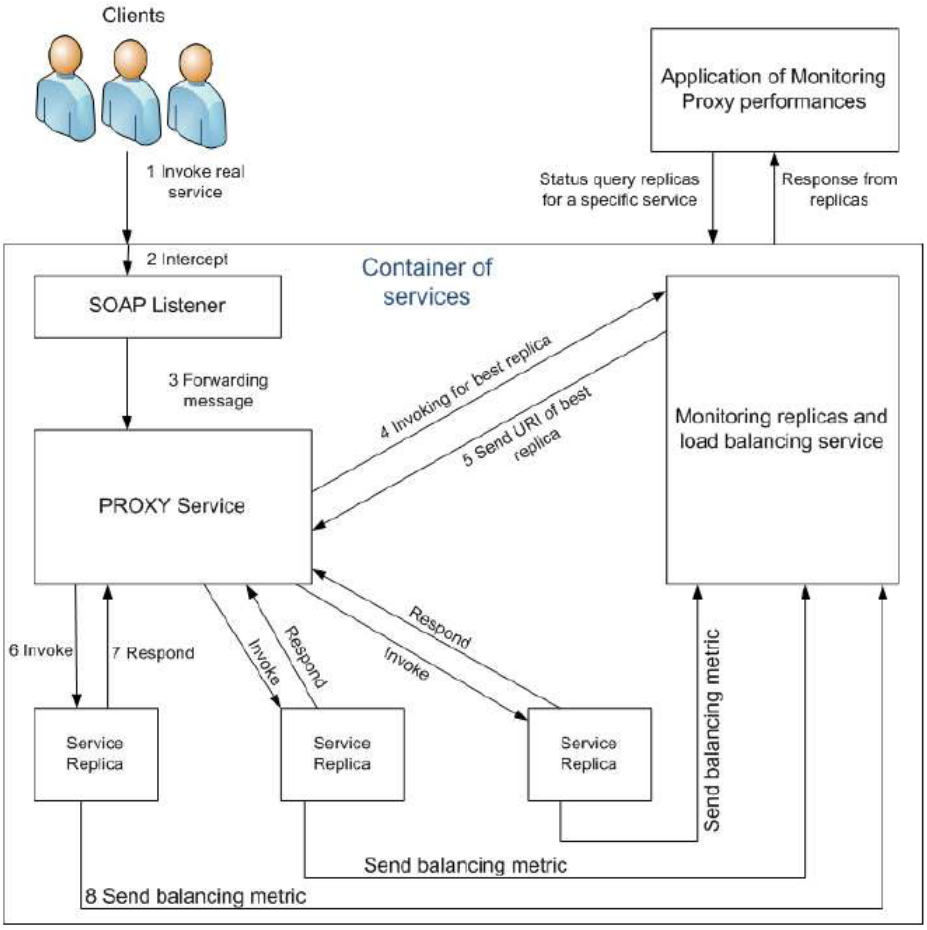

FIG. 3.1. *System Architecture*

Service (because this service is used local by providers for services deployment and monitoring) and Load-Balancing Service.

MONITORING AND LOAD-BALANCING SERVICE has the role to centralize information about replica services and to choose the best replica of a service to be invoking by Proxy Service. The best replica for a specific service is chosen according to rules described in Section 4, in order to realize a better Load-Balancing for replicas. The proposed solution is based on monitoring information from each services container. The monitoring information will be collected from container and send to a Proxy Monitor. The Proxy Monitor will calculate a balancing metric and will be capable to choose the best replica for routing processing initiated by Proxy Service.

When PROXY SERVICE receives the response submitted to a replica, it invokes a method from monitoring service for new information about this replica. Monitoring service has information about total number of requests received from the clients for a specified service, total number of processing errors transmitted to clients and total number of denied requests because of the replicas load. Concurrent access of each function to monitoring information is synchronized, to ensure exclusive access.

PROXY SERVICE has the role to receive SOAP messages from LISTENER and

to invoke replicas of service for sending a response to client and hiding eventually errors. When receive a SOAP message from Listener, Proxy Service makes a request to Monitoring and Load-Balancing Service to find the best replica of the service for which is intended the message and to signalize receiving of a new request from the client side. Received replica from this request is used for redirecting of SOAP message. If the selected replica generates an exception, then monitoring service is again interrogated to find another replica, which can be used for actual request. This algorithm is repeated a number equal of steps, if all requests generate an exception or it stops then when a request return an invalid answer. Depending of the final response of the request (valid response or exception), Proxy Service invokes the monitoring service to update local information and to send the final result to the client.

4. Load-Balancing Metric. To achieve the goal proposed in this paper, making reliable services, we implemented in each container a method to calculate a metric which help to assure load-balancing and fault tolerance. Load-balancing functionality of replicas optimizes resources utilization and minimizes the response time of a request.

4.1. Balancing Metric. Each replica of a service has a balancing level calculated according to information gathered by monitoring tool. The balancing metric will be a value between 0 and 100. 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 considers the number of threads in the system. *Live threads* represents the number of threads which are alive (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 shows the proccessors usage in the system, considering the Java Virtual Machine. *Number of processors* represents total number of processors exited in system that has the services container; the term that contained this parameter is calculated dividing this number at maximum number of processors, that is a generic number, it

represents the maximum number of processors that a system can have (if 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 occupies from all processors. Maximum 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:

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

B_4 represents a way to measure how total free space we have from total memory. When this report will tend to 0, Garbage Collector (GC) 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. Operating system allocates a total number of bytes from JVM. This memory is named Commited JVM Memory, is a fix number throughout runtime. From this Commited JVM Memory, the memory scheduler from java allocates a fix percent for Max memory. Usual, this percent is 60% 70%. All resources are allocated at runtime in total memory, which at first tends to zero and then go to max memory value. When max memory value is equal with total memory value, GC will "clean" the system, removing all allocated resources which are unreferenced. If GC will remove anything, memory scheduler will allocate extra memory for Max memory and this parameter will increase (see Figure 4.1). We have the following:

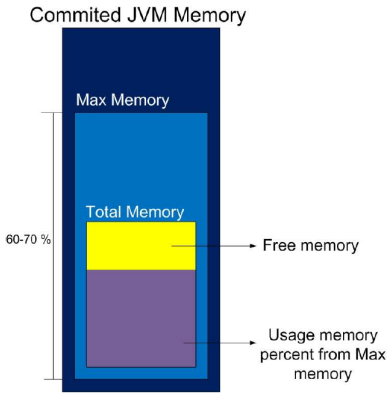

FIG. 4.1. *JVM Memory*

1. Usage memory value from total memory = Usage percent * Max memory
2. Total memory usage percent * Max memory = Free memory
3. Total memory Usage memory value from total memory = Free 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:

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

The values for weights ware considered as follows:

P_1	P_2	P_3	P_4	P_5	P_6
28	27	25	10	5	5

In the $BalancingMetric$ formula importance of weights decrease from P_1 to P_6. P_1, P_2 and P_3 are the most important weights because found formula for balancing metric wanted to represent and a trust level (P_1 and P_2), but also because operations of GC and memory scheduler have not a very height overhead in a services container context. The greater is number of requests and the smaller is number of errors, the second term of the sum will approach to P_2 value and will keep this value a period of time as long as possible. This value will decrease very low, because in the worst case at a request that finished unsuccessfully increases both the number of requests and number of errors. P_3 is also a very important weight because even if we have very few requests is enough to have just one request which lead our processors on maxim load and already the system will have a higher response time. P_4 has a smaller weight because operation of allocation memory has not a very high overhead and also P_5 and P_6 have smaller weights because term of P5 is always a constant in that container and at swap-in and swap-out operations system rarely reach.

4.2. 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.

For replica selection we proposed a dedicated service to compute reputations. The service is a centralized one, which receives information with the completion times of the jobs or information about services failures and provides a degree of reputation,

based on a metric for the QoS offered. When choosing an evaluation function for reputation calculation, a few issues there must be taken into consideration. Firstly, trust decays with time, according with the history of the quality of service provided. If, over time, the quality decreases (job execution times increase or job success rate decrease) then the function must be able to penalize it by a drop in reputation. Also, in the case of performance improvements, the reputation will grow.

This proposed solution based on reputation improves the service selection in the case of the same values for balancing metric for different replicas.

5. 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 [9] 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 scenarious:

- Empty container
- Threads Alive
- Successful Requests
- Error requests
- Best Replica
- Many Requests

5.1. Empty container. The first scenario intends to measure the balancing metric evolving in time into an empty container. No service is running on this container, only default services from Apache Tomcat and services from monitoring tool Lambda Probe. Time of measuring is 30 minutes.

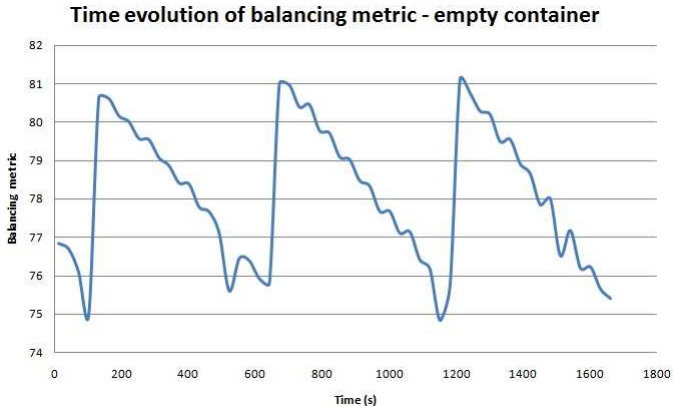

FIG. 5.1. *Time evolution of balancing metric empty container*

As is illustrated in Figure 5.1 the balancing metric varies between 75 and 81 with a periodic behavior at approximately 10 minutes. This period is due to the actions of GC which cleans references to unused objects. These actions determine the increasing of the free memory value and therefore the value of fourth parameter.

5.2. Threads Alive. The second scenario intends to measure the balancing metric evolving in time when increasing number of threads alive. Time of measuring

is approximately 15 minutes. For testing we have used services with a longer live than the period of testing because we wanted to avoid altering the requests parameter which consequently would have increased the balancing metric.

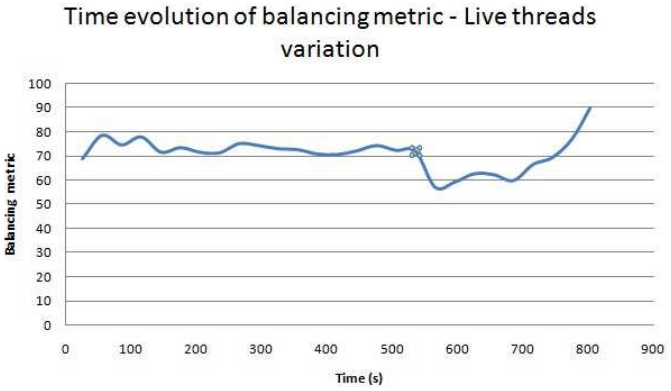

FIG. 5.2. *Time evolution of balancing metric live thread variation*

As is illustrated in Figure 5.2 the balancing metric varies between 70 and 80 before the moment of busy threads increasing. After this moment, we have increased the number of busy threads and the balancing metric decrease down to 55. After 10 minutes (600 seconds) the value of balancing metric has slowly increased but the busy threads remained the same number. The reason of balancing metric increasing is the same as in the previous test, GC has cleaned the unused references objects and possibly memory provider has allocated extra memory for total memory (value of balancing metric reached 90).

5.3. Successful Requests. This test scenario intends to measure the balancing metric evolving in time when increasing number of successful requests. Time of measuring is approximately 11 minutes. For testing we have used services with an empty method named nothing because we wanted to avoid altering the busy threads parameter which consequently would have decreased the balancing metric.

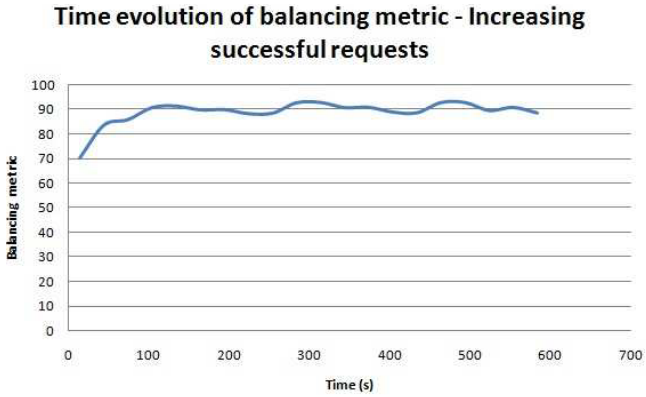

FIG. 5.3. *Time evolution of balancing metric Increasing successful requests*

As is illustrated in Figure 5.3 the balancing metric starts at 70 and increasing to

90, where is established. The second term of balancing metric represents also a trust parameter. The higher of this parameter is, the higher is trust level represented by P2 value. Default in our implementation number of requests is 1. We have made this initiation because we wanted to give half of trust to every replica. If we will have error requests (like in the next scenario) and default number of requests is 0, the second term of sum will be always 0 and the monitor service will choose at every moment first replica that has generated errors because it will not have a smaller value.

We can reach at a moment to every replica having the same value and this value cant decrease because the errors number is much higher than requests number. This means that B_2 term tends to 0. In this case we can choose the first replica, for example, even if we still send requests to it, the value of its balancing metric will not decrease, but this "problem" does not make us to choose another replica, because all other replicas are second term tended to 0, so these arent more confidence.

5.4. Error requests. This test scenario intends to measure the balancing metric evolving in time when increasing number of error requests. Time of measuring is approximately 12 minutes. For testing we have used services with a method that not exists on the container because we wanted to avoid altering the busy threads parameter which consequently would have decreased the balancing metric or other parameters like JVM CPU Usage.

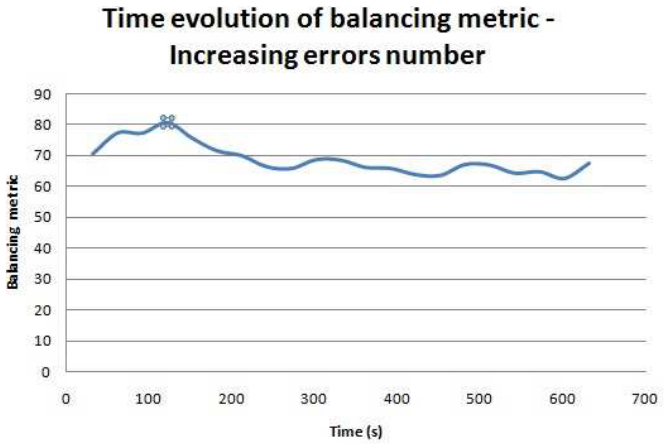

FIG. 5.4. *Time evolution of balancing metric Increasing error requests*

As is illustrated in Figure 5.4 the balancing metric starts at 70 and increasing to 80 because JVM cpu usage begins to decrease. After about 100 seconds we begin to increase the request number that finished with errors. The balancing metric starts to decrease down to 62 where begins to established because the value of second term (P2) cant decrease more and the rest of terms are constantly (only GC alters sometimes the free memory percents and balancing metric varies by 3 or 4 percents.

This parameter is part of the trusted component (P2). As we can see in Figure 5.5, number of errors represents an important role in value of trust level. If a container begins to have errors it immediately decreases the value of trust (second parameter P2) and then even if it receives a high number of request that successfully resolve them, the value of trust level increases very hard. In conclusion, is recommended that a container to have the smallest number of errors from the first moments of activity.

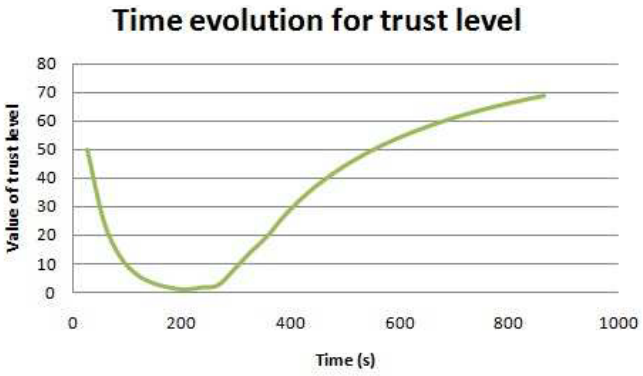

FIG. 5.5. *Evolution of trust level*

5.5. Best Replica. This test scenario intends to measure the balancing metric evolving in time in a proxy system that forwarded the received request to the best replica of the moment, with the highest value of the balancing metric.

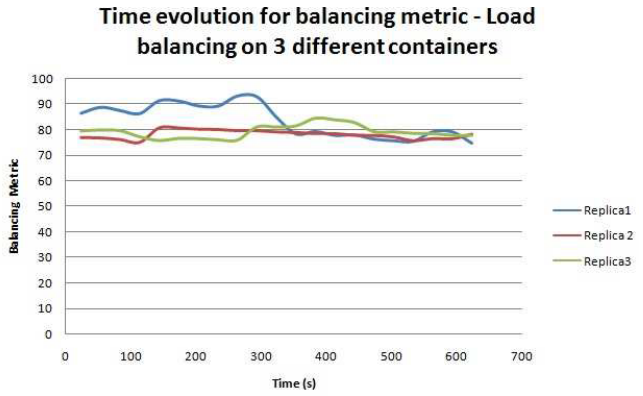

FIG. 5.6. *Evolution of trust level*

At the beginning moment replica with the best value of balancing metric is blue replica. Proxy monitor will forward to it the received request. Because blue replica finish with success all requests, it still receive requests from monitor Service. From 300 moment, blue replica finish with errors all requests and in this manner its balancing metric value decrease down to 79. In this moment replica with best value of balancing metric is green replica. It will receive all requests from monitor service and from the same reason, its value of balancing metric will increase up to 85, when at 450 (seconds) moment this value will decrease because received requests finished with errors. But the green replica still receive requests from monitor service, because it has the highest value of balancing metric (see Figure 5.6).

5.6. Many Requests. This test scenario intends to measure the balancing metric evolving in time when services container receives very many requests and its resources are tested to the fullest. As is illustrated in Figure 5.7 the balancing metric starts at 77 and increasing to 79 because JVM cpu usage begins to decrease. After about 60 seconds we begin to increase the request number that finished with errors

and the requests that requires extra resources like cpu usage and are IO intensive. After this all requests finish. This scenario repeats for two times between 250 and 350 and respectively 350 and 420 periods.

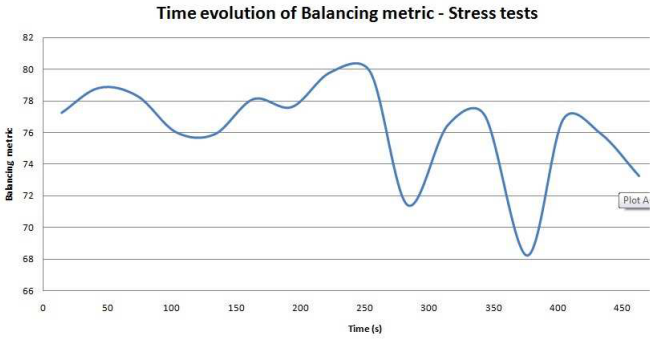

FIG. 5.7. *Balancing metric - stress tests*

6. Conclusions. As society becomes dependent by distributed systems (Grid, P2P, networkbased), it is becoming more and more imperative to engineer solutions to achieve reasonable levels of dependability for such systems. In this paper we presented an architectural approach to the development of models, methods and techniques for satisfying dependability requirements in LSDS. Dependability remains a key element in the context of application development and is by far one of the most important issues still not solved by recent research efforts.

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.

In the future we consider the possibility of extending the service implemented in several ways. First we will consider implementing a mechanism for Proxy service replication to eliminate the possibility of becoming a service insertion mechanism defects, that is to replicate and proxy service. Also, we want to integrate two services: one to calculate the administrative distance between nodes where there are services (this is a maximum flow problem) and another for detecting a fall service/container. First service can help Monitor Service to take better decisions because the balancing metric will have a more realistic value and the second service will find possible failures in the system architecture and after proxy service will be capable to replicate dynamically the invoked request. Another important aspect will be focused on reputation function. We will consider a quantitative approach of this model and we will combine the balancing metric with reputation in the process of replica selection.

Acknowledgments. The research presented in this paper is supported by national project: "SORMSYS - Resource Management Optimization in Self-Organizing Large Scale Distributes Systems", Project CNCSIS-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 "CNCSIS-IDEI" ID: 1710 (618/15.01.2009).

REFERENCES

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Vytautas. Fundamental Concepts of Dependability, 2000.

[2] J. Balasangameshwara and N. Raju. A Decentralized Recent Neighbour Load-Balancing Algorithm for Computational Grid. *International Journal*, 1.

[3] 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.

[4] G. Dobson. Using ws-bpel to implement software fault tolerance for web services. In *Software Engineering and Advanced Applications, 2006. SEAA'06. 32nd EUROMICRO Conference on*, pages 126–133. IEEE, 2006.

[5] G. Dobson, S. Hall, and I. Sommerville. A Container-Based Approach to Fault Tolerance in Service-Oriented Architectures. In *International Conference of Software Engeneering*. Citeseer, 2005.

[6] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[7] Woochul Kang and Andrew Grimshaw. Failure prediction in computational grids. In *ANSS '07: Proceedings of the 40th Annual Simulation Symposium*, pages 275–282, Washington, DC, USA, 2007. IEEE Computer Society.

[8] 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.

[9] 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.

[10] 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.

[11] Guillaume Pierre, Thorsten Schütt, Jörg Domaschka, and Massimo Coppola. Highly available and scalable grid services. In *WDDM '09: Proceedings of the Third Workshop on Dependable Distributed Data Management*, pages 18–20, New York, NY, USA, 2009. ACM.

[12] 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.

[13] Ian Sommerville and Guy Dewsbury. Dependable domestic systems design: A socio-technical approach. *Interact. Comput.*, 19(4):438–456, 2007.

[14] Michal Szymaniak, Guillaume Pierre, and Maarten van Steen. Versatile anycasting with mobile ipv6. In *AAA-IDEA '06: Proceedings of the 2nd international workshop on Advanced architectures and algorithms for internet delivery and applications*, page 2, New York, NY, USA, 2006. ACM.

[15] B. Yagoubi and M. Medebber. A load-balancing model for grid environment. In *Computer and information sciences, 2007. iscis 2007. 22nd international symposium on*, pages 1–7. IEEE, 2008.

[16] 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.

[17] X. Zhang, D. Zagorodnov, M. Hiltunen, K. Marzullo, and R. Schlichting. Fault-tolerant grid services using primarybackup: Feasibility and performance, 2004.

[18] Qian Zhu and Gagan Agrawal. Supporting fault-tolerance for time-critical events in distributed environments. *Sci. Program.*, 18:51–76, January 2010.