Web Services Resilience Evaluation using LDS Load dependent Server Models

Massimiliano Rak, Rocco Aversa, Beniamino Di Martino, Antonio Sgueglia
Dipartimento di Ingegneria dellInformazione,
Seconda Universitá di Napoli
Aversa(CE), Italy
Email: {massimiliano.rak, rocco.aversa, beniamino.dimartino}@unina2.it
antonio.sgueglia@gmail.com

***Abstract*— In the field of ICT, the term resilience is commonly used, understanding to mean the ability of a system to deliver acceptable service in the presence of faults. We interpret resilience assessment to mean assessment of ICT systems in evolving environments and conditions and in the presence of faults or failures of any kind. Resilience Measurement and benchmarking is an open problem, mainly referred to security-related problems. In this paper we propose to model Web Services (WS)as Load Dependent Servers (LDS), i.e. systems whose response time to a given request depends on the load at the time the request is received. Adoption of LDS-based models enable us to have a simple way to represent the System state. We propose to use this simple representation to quantify the system resilience comparing models built using off-line measurement and models built using on-line measurement.**

***Index Terms*— SOA, resilience, Load Dependent Server, Denial of Services**

I. Introduction

Service Oriented Computing for the development of open, large-scale interoperable systems is becoming a customary approach. Web Service standards and their implementations are widely spreading, and there are many examples of "working" solutions. In Service Oriented Application designs, abstraction layers completely hide the underlying system to all users (i.e., both to final users, system administrators and to service Developers). Due to architecture's transparency it is very hard for application/service developers to perform a quantitative evaluation at any stage of the system life cycle. This problems can be applied to any non-functional requirements(i.e. performance, security, dependability, availability, ...).

In this paper we focus on the *resilience*, using the term resilience we refer to the ability of a system to deliver acceptable service, in presence of degraded system conditions. A quantitative resilience evaluation means the definition of an index which represents the perceived quality of a system, respect to the system state. The main difference between the Fault tolerance concept and the Resilience, is that the first aims at defining techniques and tools able to build up a system able to work in presence of faults, the latter aims at quantifying the quality of a system working in fault conditions.

Manuscript received April 29, 2009; revised October 7, 2009; accepted December 1, 2009.

Resilience is referred, not only to system degraded due to system faults, but even to degradation due to security problems, as an example the presence of malicious attacks. At the state of the art '*Quantitative evaluation techniques have been mainly used to evaluate the impact of accidental faults on systems dependability, while the evaluation of security has been mainly based on qualitative evaluation criteria. Therefore, there is a need for a comprehensive modelling framework that can be used to assess the impact of accidental faults as well as malicious threats in an integrated way* [1]. As a result a lot of research work exists, aiming at resilience benchmarking.

The common approach to dependability benchmark aims at modeling the "fault loads",i.e. the loads generated by the faults. Usually the dependability benchmark takes place together with standard performance benchmark (SPEC, TPC), measuring how the performance indexes vary during as an effect of the fault loads. [1]–[3]. Following this approach both malicious threats and accidental faults are modeled in terms of a load fault, most of the papers, so, focus on the fault injection techniques and/or in building up emulated security attacks.

The main lack, in this context, is that the proposed approaches are unable to face all the "qualitative" security problems, i.e. the effect of adoption of security mechanisms, the administrative policies adopted, ... In fact the state of the art lack in terms of security evaluation metrics and security benchmarking.

The approach we propose in this paper is completely different: instead of focusing on the dependability/security benchmarks, we focus on the system: instead of using common, well-assested performance benchmarks and models, we wonder if it is possible to build up a benchmarking methodology in order to point out the system behaviour depending on its (eventually degraded) state. In other words: we aims at proposing a benchmarking technique *to discover* a performance model which relates the user perceived performances to the system state. If such a technique exists, than we will be able to evaluate in an homogeneous way, accidental faults, malicious threats, security policy and mechanisms effects, ...

As a result the approach we propose is to develop a benchmarking technique which results not in a single global index, but in a Model, i.e. a benchmarked descrip-

doi:10.4304/jcm.5.1.39-49

tion of the system, which synthesizes the obtained results and can be used to evaluate the system behaviour under given conditions.

Moreover we made the following considerations, which lead to a set of requirements for the benchmarking technique.

Performance benchmark aims at reproducing real workloads and offer global evaluation in order to have a simple way to compare two different platforms. The main lack of the common approach is that it is independent from the services: they evaluate a standard workload on a set of standard services. This approach helps the platform owners, but it is useless for service providers and final users: what happens to the target web services and workload? This consideration leads us to define the **first requirement** for our benchmarking technique: The technique should be Service-centric, i.e. we aims at measuring how a given service works. The web service platform,i.e. the containers, the server, the hardware platform, is a "parameter" for the WS evaluation. This means that the benchmarking technique accepts as an input the service WSDL.

Moreover, given a specific set of services, it is much harder to define a standard workloads: it depends on the services architecture! This means that the benchmark approach we propose must be oriented to indirect measurement: the **second requirement** is that the results should be useful for predicting the performance behaviour under real workloads.

A additional consideration should be done related to the need of quantifying effect of both malicious attacks and faults. Malicious attacks may result in Server with gradually degrading performances, i.e. the system is not in equilibrium (steady state), usually required in benchmarking. In order to catch the system behaviour in any condition, transients cannot be neglected. The **third requirement** is that the proposed benchmarking technique should be able to catch transitional behaviours.

In order to simplify the approach, as starting point we assume that the services are offered on a dedicated platform, and are, for the benchmarking, available for offline benchmarking, i.e. we are able to perform our benchmarks with the assumption that there is no additional load on the servers. After a preliminary off-line measurement we will be able to perform the same measurement on line This assumption limits the technique applicability: we cannot use it on publicly available services (such as services offered by google, viamichelin, ...), we aims at facing this kind of problems with future extensions.

The paper is organised as follows: next section illustrates the approach and describes the simple case study adopted. Section III details the benchmarking technique and related problems, using the case study services to illustrate the technique. Section IV describes the benchmarking output model, while section V illustrates some examples of the model usage and how the results can be used to evaluate the target environment and how it can be used to quantify the system resilience. Section VI illustrates the related work in the scientific community, and section VII summarizes the results and outline the future work.

II. Approach

As anticipated in introduction the approach we propose in this paper aims at defining a benchmarking technique which helps in *discovering* a performance model which relates the user perceived performances to the system state. The analysis we conducted lead us to define the following requirements:

- the technique should be Service-centric, i.e. the benchmarking technique accept as an input only the service WSDL;
- the proposed benchmarking technique should be able to catch transitional behaviours;
- the results should be useful for predicting the performance behaviour under real workloads.

Moreover the performance index we will focus on is the service response time, which is directly related to user-perceived performances.

The main assumption we made in order to build up such a technique is that the Web services can be modeled as Load Dependent Servers (LDS), i.e a WS response time received at instant `t` depends on the number (and kind) of requests that the WS platform is elaborating at instant `t`. We will demonstrate this assumption in section III. Thanks to this behaviour we are able to model Web services response time in terms of : $RT(request) = f(ServiceState, request)$, i.e the response time will be represented as a function of the service state (in terms of number and kind of pending requests) and the actual request. This approach introduces the concept of *Service State*: it depends only on the service usage and affect the performance of the service itself. We will build up the relationship *Service State-Service RT* performing a set of ad hoc measurement, performed in a dedicated environment. The result is that the services platform (hardware, operating system, middle-ware) will be taken into account by the derived model. Any kind of platform degradation will result in changes between the expected and the measured behaviour.

The proposed approach open the following questions, which are strictly related each other:

- Are Web Services Load Dependent Servers?
- How to represent *Service State*?
- How to represent (model) the relationship *Service State-Service RT*
- How to measure them?

It is very hard to reply to this question in abstract terms, so we will focus on a simple reference case study, which will help us to show the approach and validate it.

A. Test-bed and case study

As case study, we have chosen an application for remote files archiving where the files may contain confidential data. Even if the application is relatively simple, it

carries out complex tasks (analysis of large log files) and has to meet desired security requirements, according to user credentials and data confidentiality. The application is composed of a set of services to store and to elaborate files, allowing the adoption of authentication mechanisms. The services enforce different security policies and implement different security mechanisms. According to the definition given previously of security level, we can say that the service can operate at different security levels. In other words, it is possible to search for, to access and to use a service with different security features depending on the user Requested Policy (RP). The RP contains information on the credentials the user has, on the confidentiality of the log file data and/or on other service level constraints.

Figure 1 describes the service-based application, without any security feature. The application sends the log file to a *storage service*, which returns an unique *FileID*. Every time that the application invokes the *elaboration services* in order to "evaluate" the log file (e.g., to search for DoS attacks, to search for statistics on a given IP address, and so on), it passes the *FileID* to the elaboration services. These access the original file, analyse it and return the results. The proposed approach were validated implementing a framework and applying it in a dedicated test-bed. The testing environment we adopted is *Callisto*, an IBM cluster available at Second University of Naples composed of 40 nodes (IBM e326m, 2 Opteron 2.2 GHz Dual Core - RAM 2 GB - HD 2x72 GB - Network: 2 GB Ethernet), and a Front-end (IBM x346, 2 Xeon 2.2 GHz Dual Core HT - RAM 4 GB - HD RAID 6x72GB - Network: 2 GB Ethernet), managed by means of a Rocks Linux Cluster Distribution. We used the cluster nodes both as clients and as hosts for the application servers. As Service Platform we adopted three different application server: `AXIS` and `TOMCAT`, using `Geronimo` as container, `Glassfish` and `Jetty`.

As web service case of study, we developed a simple application that offers a storage service. First the web service receives the content of a plain text file, then it stores the file into the local file system. Finally the name of the just created file is returned to the service requestor. Figure 1 illustrates a simple sequence diagram in which the service usage is illustrated. Even if this application is relatively simple, it uses many different kind of resource of the service platform: disk access, memory and CPU usage, and so forth. The Storage service performance (response time) strongly depends on the underlying platform, on which no assumption has been done about (i.e. we will never use information given by the underlying system).

In order to validate the approach, as simplifying assumption, we assume that the resources assigned to the platform never changes,i.e. the only workload on the platform is given by our services. This assumption is acceptable, because in real environment the platform usually are dedicated resources, they are often built upon a virtualized hardware. Moreover we will focus on behaviour of a single service, we will analyse composed services behaviour and the effect of external workloads in future works.

III. The Measurement Technique

The core of the approach is the measurement technique, that we have defined and implemented in a framework (`SoaBNC`). As first step the measurement technique needs to represent the service state in terms of parameters which the external workload is able to control: when a request is received by a server at a time *t*, its service time depends both on the number of requests the server is actually managing at time *t*, and on the service parameters (in the storage example the service parameter is the size of file to store).

We build up our Test Clients (TCs) in order to "force" the target system in a given state, maintaining a fixed number of *Pending* requests on the server under test. The TC, as shown in Figure 5, is composed of a large set of threads, each one submits its requests to the server in a closed loop way: when the server replies, it submits the same request again for a given number of times. By increasing the number of threads inside the TC, we increase the total number of concurrent requests, and we are able to control the total amount of requests a server has to reply simultaneously at any time. TC registers the time at which each request starts (*Start* Time), the time each reply is received (*Stop* Time) and the number of pending requests, at *Start* Time and *Stop* Time.

The TC aims at generating a fix and stable load, however some response time do not respect the chosen criteria (e.g., the response time to the first (or last) messages). In order to face this problem we filter the `log` files containing the service response times. We introduce the following definition:

- **Definition:** `PN(t)` (Number of Pending Request at time t) is the number of requests at time t that the testing system has submitted to the server, without receiving a reply.

The response time of requests started at time t are interesting if and only $PN(t) = N$, where N is the number of request we are interested on. Measurement are not so accurate to grant that the PN(t) are exactly the number of requests on the server (for example some of the replies may be on the network) so we relax this condition assuming that $(PN(t)/PNexpected) > Paccuracy$ where $Paccuracy$ is the accuracy requested in order to consider acceptable the system state measurement. In the following we assumed a $Paccuracy = 0.99$, however the parameter is configurable in the `SoaBNC` framework.

The number of Pending requests can be evaluated in two ways (our framework supports both the procedures): (a) during the measurement the client take trace of the number of pending requests; (b) analysing the log file with start and stop time of each request. In the following measurements we adopted the first approach, the second one is, usually, more time expensive.

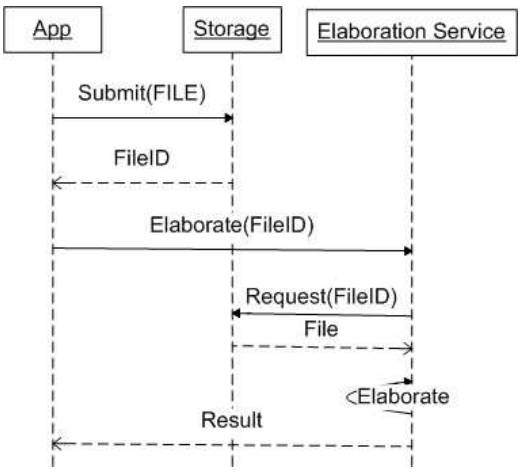


Figure 1. Log Analysis Service Usage

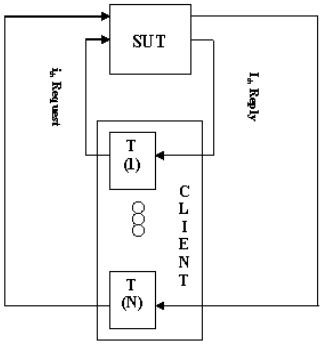


Figure 2. The Closed Loop approach

In order to graphically evaluate a state measurement, we adopt three different kind of plots:

- **Pending Plot**: for any given time of the experiment (shown on the x-axis) it reports on the y-axis the number of *Pending* requests($PN(t)$); this plot helps us to find the *window* of messages (and their starting times) to be evaluated.
- **Response Time Evaluation** (RTE): reports on x-axis the time in which a valid message (messages started in the valid window) has been sent and on the y-axis its evaluated response time.
- **Response Time Histogram** (RTH): reports the number of messages (normalised to one) which have the response time in an interval of values of fixed length (the interval we use in the examples is $MaxResponsetime/20$, which has shown us an acceptable resolution); this plot helps us to graphically evaluate the statistical distribution of the response time.

Figure 3 shows an example of the plots (side a *Pending* Plot, side b *RTE* plot, side c *RTH* plot), for the case in which we have 60 concurrent requests for the storage server with a storage file of an hundred of bytes.

A. The Measurement Procedure

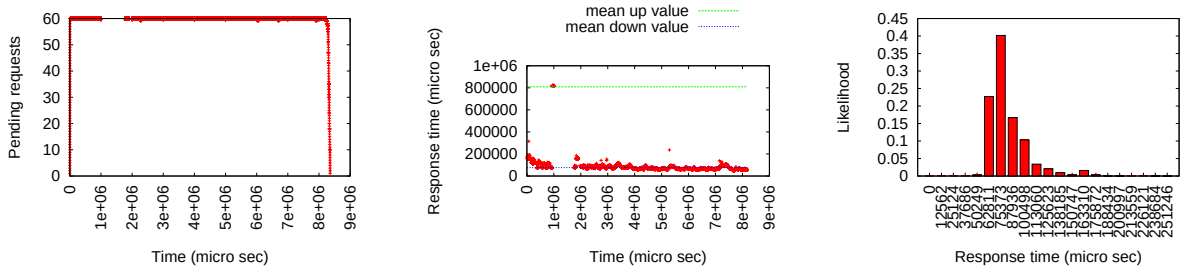
The proposed Measurement Technique, for the performance characterisation of a given service S, can be summarised in the following steps:

1) Define the Service Parameter (SP) values interval, i.e., in the case of the `storage` services it is the interval of values the file dimension has (in our experiments 10, 100, 1000 bytes).
2) Define the Concurrent Request (CR) values Interval, i.e., the number of concurrent requests we are interesting to evaluate to build our performance model
3) Start-up the Tests, at the state we use a Full Factorial Design, evaluating all the combination of SPs and CRs
4) Build up a report containing three plot (Pending, Response Time Evaluation, Response Time Histogram) for each test set.

The set of results will be adopted (section IV) to build up models able to compare service platforms and predict the service response time under any given workload.

B. Measurement Technique Validation

In order to validate the approach we must show that the System Response Time (i.e. the response time of the



(a) (b) (c)

Figure 3. Evaluation Plots: (a) Pending Plot, (b) Response Time Plot, (c) Histogram Plot

target service on the SUT) depends on the number of Pending requests. This means that the resulting values should be a constant value. Due to measurement error, we have to take into account that the response time has a statistical form, so we expect that it should have a Normal Distribution, whose mean value represents our constant. Standard deviation will help us to understand the accuracy of our measurement.

Due to space limits, in this paper we will illustrate the validation procedure only on the `Storage` service (even if we made similar consideration on a larger set of services and on their compositions), described in detail in section II-A, which accepts as a parameter a file and store it on the server file system. We modeled its behaviour using three different file dimension (10 bytes, 100 bytes , 1000 bytes) and varying the number of concurrent request between 5 and 200 (step 5). Figure 4 shows some of the results obtained, we just selected a set of values from the full set of results, the first column contains the Pending Plot, the second the RTE plot, while the last the RTH plot. All the result refer to a file dimension of 100 bytes, the first row contains the plots obtained for 40 concurrent requests, the second for 100 requests and the last one for 140 requests.

Some consideration are really useful at this step. The first one is that the Pending plot shows that the system not always is able to maintain the stable load on the server (sometimes the client becomes too slow, for example due to page faults or other actions on the client host machine). The Filtering technique let us to find the windows in which the values are of interest. Moreover not always the response time is a single constant value, for 40 and 100 concurrent requests, we obtain two different values. It is interesting to point out that, as show by the histogram (RTH plot) both have a Normal distribution: the server has two different behaviour! We will call this the system *Double Response Time*. Moreover it is important to point out that the two measured behaviour have no relation to the *Pending* plot shapes, in fact they are distributed on all the response in all the valid windows. This result has two effect on our methodology: the first one (negative) shows that the system has an its own evolution that is independent from the workload we submit to it, the second one (positive) is that this evolution is regular (at the end we have just two different values) and can be statistically modeled using the proposed approach! Note that when the number of the concurrent request grows the results become more and more stable (from 140 concurrent requests to more the result is stable). Moreover the lower value is usually more stable than the higher value (i.e. the standard deviation is lower).

We suppose (but this analysis is unproven, and out of the scope of this paper) that the second behaviour (the one with higher response time) depends on the memory usage: it is the response time of service when the garbage collector of the JVM under tomcat starts. This will justify the higher variance of this result.

In order to complete the analysis we have to point out that some outliers are registered, as shown in the RTE plots, in most of the case this happens just before the client becomes unable to maintain the load on the server. This can be a third server behaviour (whose response time become so slow that the client has some side effects). In this second case, we suppose that it depends on another cyclic phenomenon on the server, for example it could be the effect of page fault.

Our conclusions are that the proposed technique is able to force the system states, in which the system is not completely stable, but whose behaviour can be easily modeled (as we will see in the following sections). Moreover the proposed approach help us to catch an important behaviour typical of this class of systems, and which characterises the underlying platform. So the proposed technique can be adopted, in future, to evaluate and directly compare the service platforms, independently from the offered services.

IV. Benchmark Result

The proposed measurement technique produces a large set of values, which should be collect in a clear and understandable way. Instead of proposing a single performance index, we collected all the result in a synthetic representation, named *Basic Model (BM)*. BM is a simple way to collect the measurement data obtained from the technique above illustrated, it represents the system as a simple Response Time vs (Pending Requests,Service Parameter) table, containing all the measurement results, it can be seen as a tabular description of the function: $RT = f(ConcurrentRequests, ServiceLoad)$. Table I illustrates the Basic Model representing the Storage Service in our test-bed environment. The BM can be used to build up predictive models, it contains all the needed

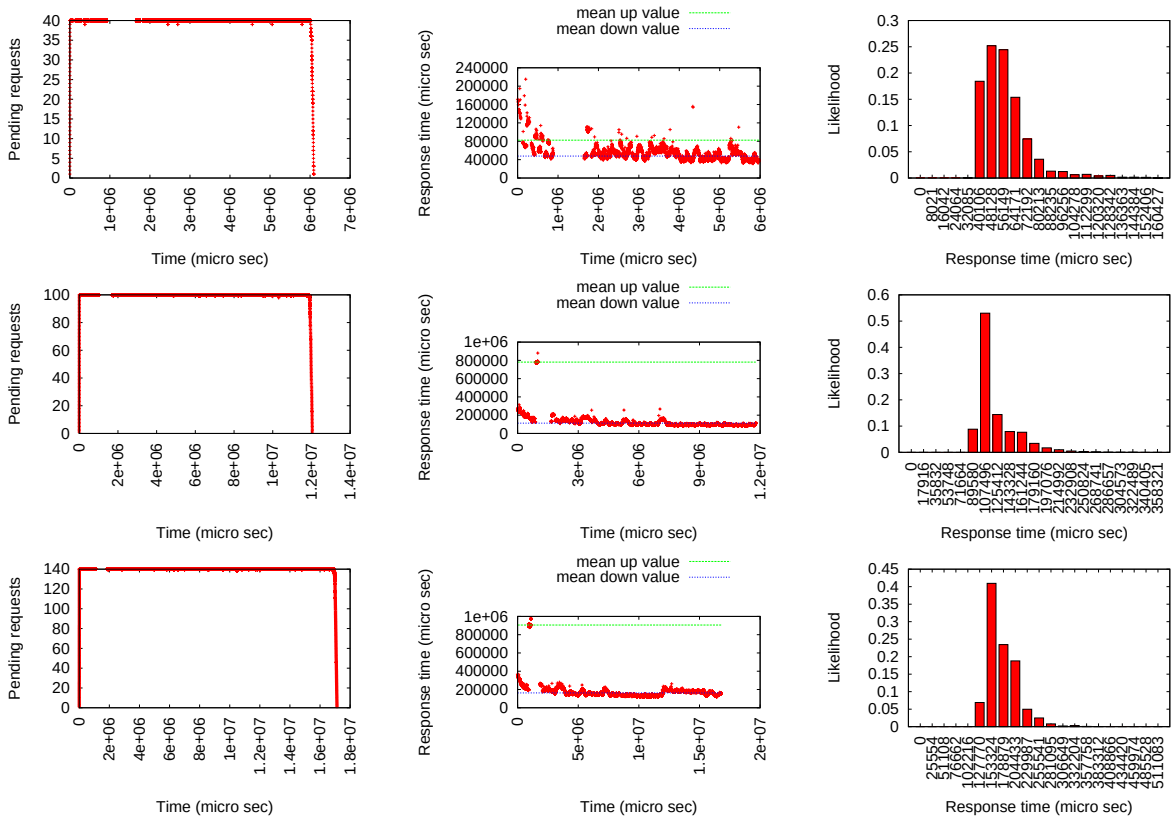


Figure 4. Storage Service Evaluation Plots, File Dimension=100 byte, column (a) Pending, column (b) RTE,column (b) RTH; row 1) 40 concurrent requests, row 2) 100 concurrent requests, row 3) 140 concurrent requests

information to build up a regression model or to build up a simulator, it is out of the scope of this paper to details the predictive models. In order to build the table we need to face two main problem: designing the set of experiment we want to perform and collect and managing problems like the double response time values. About the former problem, we have a given Service to characterise, and the measurement technique let us to model it in terms of 1) the concurrent requests, 2) the service parameters, we just adopt a Full Factorial Design, collecting all the possible combination of variable values. The double value response time can be modeled assigning a probability to each mean response time, to obtain this values we adopted the following procedure:

1) analysing the RTH plot we are able to find a threshold between the two normal distributions;
2) we evaluate the total number of messages in the valid window ($Total$)
3) we evaluate the number of messages whose response time is under/upper the threshold ($Low/High$)
4) we evaluate the probability the response time is under (up) the threshold as : $Low/Total$ ($High/Total$)

We call First Local Mean Response Time (LMRT1) the mean response time for all the values under the threshold and First Local Mean Probability (LMP1) the probability that the response time is under the threshold. Moreover we will use the name LMRT2 and LMP2 (Second Local Mean Response Time and Second Local Mean Probability) for the values up the threshold. At the state of the art we never meet system with three different RT, in that case the proposed approach can be easily extended. As a result we obtain the following table

Note that the proposed table can be used in order to build up Regression Models of the target system, being derived from a Full Factorial Design in which PR and SP are the primary factors and the response time is the response variable and in the section IV-A it is explained how this models can be created.

In order to consult the BM table we propose a set of *BM Views*, i.e. plots which summarise the most useful information extracted from the measurement. This plot can be easily generated from the BM table. that reports less information, but helps in the system analysis. The first two kind of plots (we call the the *Summary* plots) need an additional performance index: Global Response Time (GRT) which is the mean response time, it can be easily evaluated as: $GRT = p1 * v1 + p2 * v2$, they are:

- **Summary Service Parameter Variation Plot (SSPV plot)** it is an histogram plot, in which we report on x-axis the Service Parameter values and on the y-axis the GRT for each Pending Request).
- **Summary Pending Requests Variation Plot (SPRV plot)** it is an histogram plot, in which we report on x-axis the Pending Request values and on the y-axis

Pending	100 byte		1000 byte		10000 byte	
10	20778.138889	0.207852%	23759.088123	0.360000%	230474.045455	0.712813%
	7417.747813	0.792148%	8932.946121	0.640000%	178053.461538	0.287187%
30	20064837.100000	0.113208%	20016567.155405	0.128139%	496995.153846	0.115044%
	13193.502128	0.886792%	9251.671301	0.871861%	104890.650000	0.884956%
60	39889774.212389	0.499915%	39896118.759369	0.488452%	38351145.329820	0.500000%
	20105687.837700	0.500085%	19404894.255518	0.511548%	21832269.444747	0.500000%
90	59723169.181102	0.245397%	59824112.668784	0.250170%	57668507.421627	0.249631%
	39306771.734448	0.754603%	40057044.840297	0.749830%	40938348.972189	0.750369%
120	59995541.706524	0.997957%	59998405.808941	0.991121%	60155694.951018	0.998894%
	40157000.916667	0.002043%	22146599.894231	0.008879%	42366593.000000	0.001106%
150	80008628.591489	0.544572%	79875403.686498	0.545579%	78400065.079701	0.548427%
	59510484.347750	0.455428%	59850399.971206	0.454421%	62401520.353851	0.451573%

TABLE I.
THE STORAGE SERVICE BASIC MODEL TABLE

the GRT for each Service Parameter.

Together with the summary plots we propose three more detailed plots (we call them *Stochastic* plots), which gives a graphical representation of the proposed BM.

- **Local Mean Service Parameter Variation Plot (LMSPV plot)** it is a line plot, in which we report on x-axis the Service Parameter values and on the y-axis two lines, one containing the LMRT1 values and the second the LMRT2 values for each Pending Request.
- **Local Mean Pending Requests Variation Plot (LMPRV plot)** it is a line plot, in which we report on x-axis the Service Parameter values and on the y-axis two lines, one containing the LMRT1 values and the second the LMRT2 values for each Service Parameter.
- **Mean Probability Service Parameter Variation Plot (LMSPV plot)** it is a line plot, in which we report on x-axis the Service Parameter values and on the y-axis two lines, one containing the LMP1 values and the second the LMP2 values for each Pending Request.
- **Mean Probability Pending Requests Variation Plot (LMPRV plot)** it is a line plot, in which we report on x-axis the Service Parameter values and on the y-axis two lines, one containing the LMP1 values and the second the LMP2 values for each Service Parameter.

These plots help in understanding how the system behaviour evolves and can be used in order to build up predictive models, as shown in the following section.

A. The Regression Model

In this subsection it is described how build up simple and multiple regression models using the results obtained in the previous chapter. In this case the number of concurrent requests cr and the request parameter rp are the primary factors and the mean response time y is the response variable. The linear multiple regression model is expressed by the equation:

$$y = b_0 + b_1 * cr + b_2 * rp + e \quad (1)$$

The terms e represents the error of the model. If we organize the results of the experiments in a matrix where y_i is the mean response time of the i-th couple of (cr_i, rp_i) and cr_i and x_{rp_i} are the values of the number of pending requests and the request parameter for that couple, we obtain that:

$$\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & cr_1 & rp_1 \\ 1 & cr_2 & rp_2 \\ \dots & \dots & \dots \\ 1 & cr_n & rp_n \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} + \begin{bmatrix} e_0 \\ e_1 \\ \dots \\ e_k \end{bmatrix} \quad (2)$$

The previous expression can be written as $\mathbf{Y} = \mathbf{XB} + \mathbf{e}$. The values of b_0, b_1 and b_2 that minimizes the Sum of Squared Error are [4]:

$$B = (X^T X)^{-1}(X^T y) \quad (3)$$

Where $B = [b_0 b_1 b_2]^T$. The following table summarizes some parameter characterizing the model.

Parameter	Value
Model	**y = Xb + e**
b	$(X^T X)^{-1}(X^T y)$
y_{mean}	$\frac{\sum_{i=1}^{n} y}{n}$
SS0	$n(y_{mean})^2$
SSY	$\sum_{i=1}^{n} y_i^2$
SST	$SSY - SS0$
SSR	$SST - SSE$
R^2	$\frac{SSR}{SST}$

The **coefficient of determination (R^2)** indicates the percentage of the variance of the model that is related to the regression. The higher the value of R^2, the better the regression.

If we apply this technique to the storage service we obtain that:

$$B = [-6.9128 * 10^{06} 4.939910^{05} 6.0008 * 10^{03}] \quad (4)$$

and $R^2 = 0.72533$. In order to have a more accurate model, the previous one can be linearized by fixing the value of request parameter (linear simple regression model):

$$y = b_0 + b_1 * cr \quad (5)$$

In this case the whole system is described by 3 functions (one for each value of request parameter). For instance for the storage service by fixing the value of request

parameter at 100 the system can be modelled by the function 6.

$$y = -5135804 + 539528 * cr \quad (6)$$

In this case the value of the coefficient of R^2 is 0.98 and the fig. IV-A shows the relation between the simple linear regression model and the mean response time.

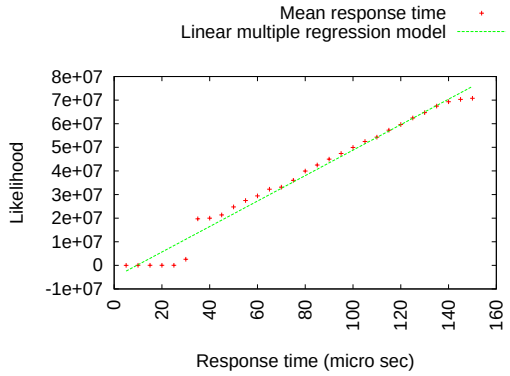


Figure 5. Simple linear regression model

V. Resilience Evaluation with LDS models

The BM is a powerful tool for comparing system behaviours, in fact it is a tabular representation of the system states. In this section we will show how, adopting the proposed technique, we are able to offer a clear quantification of resilience when degradations are affecting the system (and in consequence the BM). Moreover the BM helps in isolating the causes of the service quality degradation.

A. System Configuration

As first example we compare the measurement after a system degradation. We assumed that the underlying web services platforms (i.e. the hardware, the operating system and the application server) are offered as black box, however their configuration (and their degradations) have direct effect on the offered services performance: in this example we assume that the platform owners change a parameter (the number of threads in the pool for the application server), and we will show how this change directly affects the BM.

Figures 6 and 7 show the plots SSPV (on side a) and SPRV (on side b) for the storage service. As shown in the figure, even if these plots are just a summary, they can be really interesting and they show that our approach is able to put in evidence details about the system architecture, in fact if we focus on figure 7 we can notice that the service response has two different behaviour: one before and one after a threshold (in our case it is 115 Pending Requests) that differ in term of order of magnitude of the mean response time.

In order to put in evidence this behavior, we emulated a system degradation, comparing the results for two the Application Server thread pool dimension: 80, whose results are described in figure 6, and 150, whose results are described in figure 7. It is interesting to note that the threshold for the behavior change does not correspond to the maximum value, but to a value that is a few little: (115 for a limit of 150 and 35 for a limit of 80). Moreover both before and after the threshold the behavior are linear.

Figure 8 contains the four stochastic plots for the Storage service. Note how at the Pending threshold value (the one at which the RT mean value changes) we have an inversion of probability between low and high values.

The proposed example put in evidence that the proposed benchmarks, together with the resulting model, offer a powerful tool for comparing different platforms states from the point of view of service performances.

B. Service behaviour changes during a DoS attack

As second example we emulated a simple DoS attack: a client generates a continuous flood of service invocations, but it never waits for the service completion. The result is that the application server receives a large number of uncompleted requests.

We started up the attack emulator and, during the attack, we started up our benchmarks.

The attack we choose to emulate is very simple and has only a little effect on the server: it represent an initial DoS that will improve in time. We aims at showing that the LDS model changes even in this case, helping in identifying the problem.

We compare the result in two different condition: *off-line*, i.e. the benchmark takes place in ad-hoc environment, and *on-line*, i.e. we started up the attack emulator and, during the attack, we started up our benchmarks.

In order to reduce the benchmark time the on-line measurements are focused on a limited set of concurrent requests (up to 50, while offline measurement offer measures up to 150).

As anticipated the response time in both of the cases is similar, but the stochastic plots show the difference between the two different tests, as shown in figure 9, while the probability of up and down values in common condition continuously changes between in the off-line measurement, the presence of the DOS completely changes the plots.
The new behaviour implies that the probability of obtaining the better response time (the low value) is always less than the second response value (the worst value). When the attack starts, this result does not affect the final mean response time, but when the attack increases the performance loss will be visible.

VI. Related Work

Evaluation and comparison of Web Services (and web servers) platform is a common problem, usually faced , in business world, with the typical benchmarking approach: the platform performances are characterised by a performance index evaluated on a standard workload. SPECweb05 and TPC-App are the most known standard

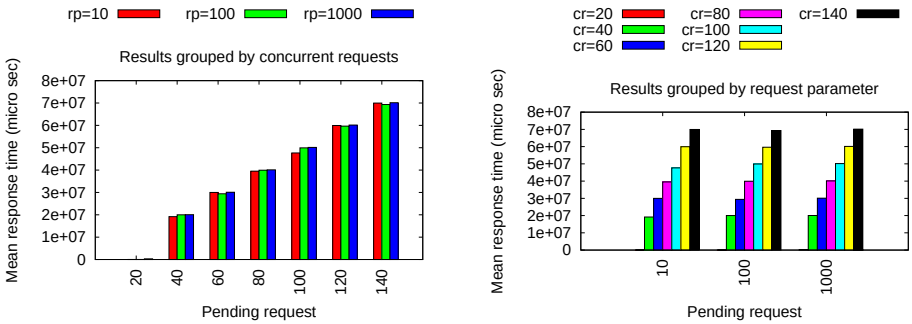


Figure 6. Summary Plots: Thread limit 80 (a) Service Parameter Variation Plot (SSPV) (b) Pending Requests Variation Plot (SPRV)

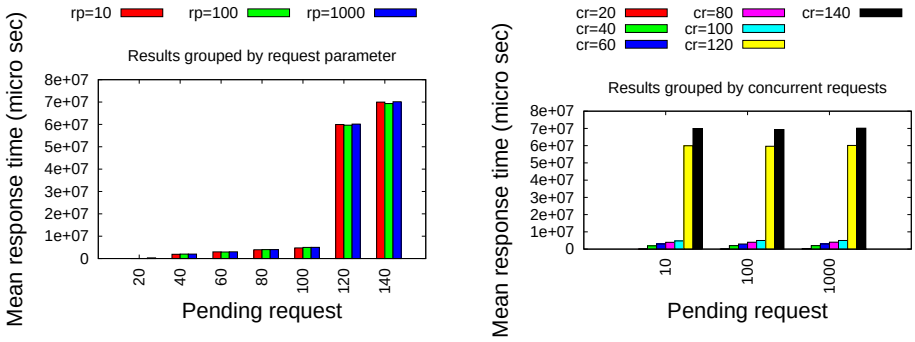


Figure 7. The Thread Pool Effects: Thread limit 150 (a) Service Parameter Variation Plot (SSPV) (b) Pending Requests Variation Plot (SPRV)

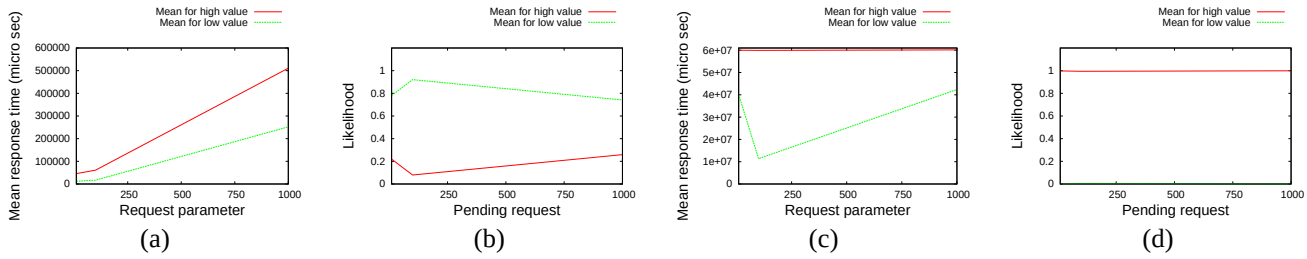


Figure 8. The Stochastic Plots: (a) Local Mean Service Parameter Variation Plot (LMSPV) (b)Local Mean Pending Requests Variation Plot (LMPRV) (c)Mean Probability Service Parameter Variation Plot (LMSPV) (d)Mean Probability Pending Requests Variation Plot (LMPRV plot)

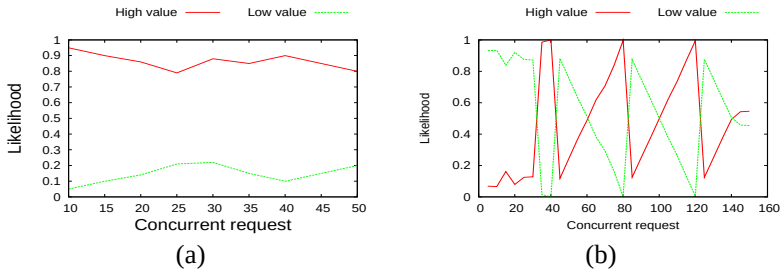


Figure 9. The Stochastic Plots: (a) With DoS; (b) Without DoS

benchmarks. Benchmarking approach aims at evaluating the server under a set of well-known workloads and to compare different platforms respect to use cases considered typical. They can be used for high level comparison of platforms (i.e. to sell a product), but they are of little use in application (services) tuning and optimization and for predicting the platform behaviour under different and user-defined workloads.

It is out of the scope of this paper (and this section) to offer a complete survey on the performance prediction and modeling of Web services platforms, we will report only that models and techniques that, at the best of authors knowledge, have common features with our proposal. A complete survey on model-based performance models is [5], while [6], [7] are interesting readings which exploit the main active research activities on Software Performance Engineering (SPE).

As pointed out in [5], many modeling techniques and formalism exist and are applicable to performance prediction, between the extended queue network modeling technologies, the LQN, as proposed in [8] may be compatible with the proposed approach. In [9] (in

figure 12) the authors perform some tests repeating the same calls a fixed number of times (as we do) and in fact they have a behaviour similar to the our ones (the double value response time, due to the garbage collection in their analysis) during the validation steps. They are interested in the system start-up, while we focus on the stable behaviour. Our works try to build up a rigorous approach to face that kind of problems.

Wu and Woodside in [10], illustrate a framework which combines measurement and model, in their approach the workload is driven by user profiles and on real system usage. Measurement calibrate the models, while the models define and manage tests. The main difference with our approach is that our measurement defines (discover) the model and we use user-independent workloads (which we consider in the simulated predictive model).

The concept of Load Dependent Servers, proposed in [11] (the idea originated in [12], [13]), perfectly match with our modeling approach, they use standard benchmark (as workload) and regression models to represent the system behaviour, proposing a multi step methodology for model calibration. The main differences are the way in which their models are built and characterised. The approach proposed in this paper can be used as an alternative way to build up their LDS regression models.

Mathur and Apte, in [14] follow the idea of load-dependent behaviour, modeled by LDS, and study the evolution of LDS, modeling them using queue networks. Note that they focus on simulation analysis, they use measurements only to validate their models.

A completely different approach to black box system modeling, that should be considered, is the adoption of control-theoretic feedback loops, some interesting examples of their application to performance of computer systems are [15]–[18].

VII. Conclusions and Future Work

In this paper we proposes a benchmarking technique, which helps both in understanding the behaviour of the hosting platform and in quantifying the system resilience. The approach we adopted in building up the measurement technique was to define a target system category (Load dependent servers) and *force* them in a given state, then we derive (in an automated way) the models from the measurement. The measurement technique adopted, at the best of the authors knowledge, even if sporadically adopted, were never defined in this way, and never adopted in order to build up automatic performance model.

The resulting models were used to compare the services behavior in degraded server conditions, pointing in evidence how the benchmarks catch the system degradation and helps in defining their causes.

In our humble opinion a relevant result, obtained applying the technique on the proposed case study, is the discovering of the LDS multiple behaviour (the double response time). It is due to the presence of cyclic evolutions of the server (garbage collections, effects of the memory hierarchy, ...) and they are (partially) independent from the workload the server is subject to. This behaviour, in complex system, may be really hard to discover, but has a visible effect upon the overall performance evaluation of the target system (the delay introduced sometimes are many times bigger than the most common response time). This behaviour is very common, but very hard to discover and reproduce.

The next step in the research activity is to use the technique and the models we introduced to create simulation models able to predict the system behavior under a given workload and in presence of faults. We aims at adopting the simulation both in the development stages, in order to predict how the degradation of a single service could affect the behavior of composed services, and at run time, predicting the system evolution under known workloads in order to optimally tune the system. Moreover we aims at investigating more in detail which are the kind of systems that can be modeled following the proposed approach, besides web services.

Acknowledgment

We wish to thank prof. Villano and the anonymous reviewers for their comments, which helped us in improve the paper quality.

References

[1] A. van Moorsel *et al.*, "State of the art, Tech. Rep. D2.1.a, February 2009.

[2] B. M. L. S. Don Wilson, "Progress on defining standardized classes for comparing the dependability of computer systems." unknown, 2002. [Online]. Available: http://citeseer.ist.psu.edu/570755.html

[3] H. Madeira, "Assessing, measuring, and benchmarking dependability and resilience," in *LADC*, ser. Lecture Notes in Computer Science, A. Bondavalli, F. V. Brasileiro, and S. Rajsbaum, Eds., vol. 4746. Springer, 2007, p. 238.

[4] R. Jain, Ed., *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-InterScience, Apr 1991.

[5] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: A survey," *IEEE Trans. Software Eng.*, vol. 30, no. 5, pp. 295–310, 2004.

[6] C. M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *FOSE*, L. C. Briand and A. L. Wolf, Eds., 2007, pp. 171–187.

[7] C. M. Woodside, "The relationship of performance models to data," in *SIPEW*, ser. Lecture Notes in Computer Science, S. Kounev, I. Gorton, and K. Sachs, Eds., vol. 5119. Springer, 2008, pp. 9–28.

[8] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar, "The stochastic rendezvous network model for performance of synchronous client-server-like distributed software," *IEEE Trans. Computers*, vol. 44, no. 1, pp. 20–34, 1995.

[9] J. Xu, A. Oufimtsev, C. M. Woodside, and L. Murphy, "Performance modeling and prediction of enterprise javabeans with layered queuing network templates," *ACM SIGSOFT Software Engineering Notes*, vol. 31, no. 2, 2006.

[10] X. Wu and C. M. Woodside, "A calibration framework for capturing and calibrating software performance models," in *EPEW*, ser. Lecture Notes in Computer Science, N. Thomas and C. Juiz, Eds., vol. 5261. Springer, 2008, pp. 32–47.
[11] M. Curiel and R. Puigjaner, "Using load dependent servers to reduce the complexity of large client-server simulation models," in *Performance Engineering*, ser. Lecture Notes in Computer Science, R. R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz, Eds., vol. 2047. Springer, 2001, pp. 131–147.
[12] J. Zahorjan and E. D. Lazowska, "Incorporating load dependent servers in approximate mean value analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 12, no. 3, pp. 52–62, 1984.
[13] H. Perros, Y. Dallery, and G. Pujolle, "Analysis of a queueing network model with class dependent window flow control," *INFOCOM '92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE*, pp. 968–977 vol.2, May 1992.
[14] V. Mathur and V. Apte, "A computational complexity-aware model for performance analysis of software servers," in *MASCOTS*, D. DeGroot, P. G. Harrison, H. A. G. Wijshoff, and Z. Segall, Eds. IEEE Computer Society, 2004, pp. 537–544.
[15] M. Karlsson and M. Covell, "Dynamic black-box performance model estimation for self-tuning regulators," in *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 172–182.
[16] J. L. Hellerstein, Y. Diao, S. Parek, and D. M. Tilnury, *Feedback Control of COmputing Systems*. Wiley-IEEE Press, 2004.
[17] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 17, no. 9, pp. 1014–1027, Sept. 2006.
[18] T. Abdelzaher, K. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: a control-theoretical approach," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 1, pp. 80–96, Jan 2002.

Massimiliano Rak is an assistant professor at the Second University Napoli, Italy. His research activities include both theoretical and experimental issues, in the areas of performance evaluation of computing systems, parallel and distributed software engineering, security of information systems. He received the laurea degree in Computer Science Engineering from the University of Napoli Federico II in 1999 and Ph.D. in Computer Engineering from the Second University of Napoli in 2002.

Rocco Aversa graduated in Electronic Engineering at University of Naples sin 1989 and received his Ph.D. in Computer Science in 1994. He is Associate Professor (Assistant Professor from 1995 to 2004)in Computer Science at the Department of Information Engineering of the Second University of Naples. His research interests are in the area of parallel and distributed systems. The research themes include: the use of the mobile agents paradigm in the distributed computing; the design of simulation tools for performance analysis of parallel applications running on heterogeneous computing architectures; the project and the development of innovative middleware software to enhance the Grid computing platforms. Such scientific activity is documented on scientific journals, international and national conference proceedings Rocco Aversa participated to various research projects supported by national organizations (MURST, CNR, ASI) and in collaboration with foreign academic institutions. In 2005 he was appointed in the board of the directors of the consortium "Centro Regionale Information e Communication Technology" as the representative of the Second University of Naples.

Beniamino Di Martino is a Full Professor at the Second University of Naples (Italy). He is the author of five international books and more than 100 publications in international journals and conferences. He served as General and Program Chairman, and a member in programme committees, of several international conferences, and as Guest Editor for several journals special issues. He is an Editorial Board Member and Chair of international journals. He is Vice Chair of the Executive Board of the IEEE CS Technical Committee on Scalable Computing. His research interests include programming and compiler techniques for high-performance and grid computing, mobile and intelligent agents, automated program analysis and transformation, reverse engineering, semantic-based information retrieval, the semantic web and semantic web services.

Antonio Sgueglia is a Ph.D. student at Second University of Napoli, Italy. His research activities are focused on modeling and on performance evaluation of distributed Business to Business SOA architectures. He received the laurea degree in Computer Science Engineering from Second University of Napoli in 2003. He works as System Engineer at Ericsson Telecomunicazioni S.p.A. He worked as senior researcher at CIRA (Italian Aerospace Research Centre)