Model-based Performance Anticipation in Multi-tier Autonomic Systems: Methodology and Experiments[1]

Nabila Salmi*‡, Bruno Dillenseger*, Ahmed Harbaoui*†, and Jean-Marc Vincent†
* Orange Labs, Grenoble, France
Email: bruno.dillenseger@orange-ftgroup.com, nabila.salmi213@gmail.com
† LIG, MESCAL Project, Grenoble, France ‡ LISTIC, University of Savoie, Annecy, France
Email: {ahmed.harbaoui, jean-marc.vincent}@imag.fr

***Abstract*—This paper advocates for the introduction of performance awareness in autonomic systems. Our goal is to introduce performance prediction of a possible target configuration when a self-* feature is planning a system reconfiguration. We propose a global and partially automated process based on queues and queuing networks modelling. This process includes decomposing a distributed application into black boxes, identifying the queue model for each black box and assembling these models into a queuing network according to the candidate target configuration. Finally, performance prediction is performed either through simulation or analysis. This paper sketches the global process and focuses on the black box model identification step. This step is automated thanks to a load testing platform enhanced with a workload control loop. Model identification is based on statistical tests. The identified models are then used in performance prediction of autonomic system configurations. This paper describes the whole process through a practical experiment with a multi-tier application.**

***Keywords*-Autonomic systems; performance; automatic modelling; queuing network model; load injection**

I. Introduction

A. Autonomic computing and performance management

Management of modern distributed systems is becoming increasingly complex and costly. Autonomic computing typically addresses this issue by providing systems with self-management capabilities. A common approach to building self-managing systems has been sketched by [1], through the well-known MAPE-K control loop (Monitor, Analyze, Plan, Execute - Knowledge): some self-* features (e.g., optimization, configuration, healing and protection) are implemented in the system in the form of feedback loops that result in system reconfiguration plans to be executed when special undesired situations are met. Reconfigurations typically result in removing, adding or replacing one or several system constituents, thus resulting in a new configuration. Here, we consider changing constituent parameters (e.g., tuning) as a component replacement inasmuch its behavior changes, especially from a performance point of view.

Reconfiguring a distributed application may result in performance changes, ranging from anecdotal to dramatic. In the case of critical or Service Level Agreement-ruled systems, it may be quite relevant to evaluate the performance of a candidate new configuration before actually deploying it. This remark applies to any self-* feature-driven reconfiguration, but it particularly applies to self-optimization. This sort of feature may typically compare different candidate configurations, looking for an optimized trade-off between an expected performance level and operational constraints and costs.

[1]This work is supported by the French ANR, through the Selfware and SelfXL projects, and ANRT.

This paper deals with the introduction of a strong performance-awareness in autonomic systems, in order to drive the Analyze step of the MAPE-K loop with relevant performance Knowledge, combined with performance analysis or simulation capabilities. To do this, our approach consists in relying on performance models of a distributed application's constituents, and then composing these models according to interactions between constituents, to get a performance prediction of an application configuration. We typically address distributed applications where some constituents may be replicated in order to increase the overall application performance (e.g., multi-tier web applications). In this introduction, we sketch the global process, as presented in [2].

B. Identifying black boxes

The first roadblock we meet is getting the constituents' performance models. Applications, middleware and systems based on common information technologies typically come with poor performance-related specification, if any. At a certain granularity, the inner architecture of some constituents is either so complex or under-specified that trying to infer a performance model for each one would practically take far too much effort. However, a certain granularity of decomposition seems to be humanly affordable, at least for distributed applications. For instance, an HTTP front-end, an EJB container and a database is a straightforward level of decomposition in the context of multi-tier Java EE applications. Based on this, our approach is two-fold:

1) decompose a distributed application into constituents, called *black boxes*, with a relevant granularity,
2) automatically get a performance model of each black box through an experimental stimulus-response observation principle.

The relevant granularity level is a trade-off between the decomposition feasibility (with regard to available information and complexity) and the final model accuracy and sizing opportunities. The major criterion is sizing opportunity: if one sub-element of a black box can be replicated to increase the workload capacity of the sub-feature it supports, then there is a big motivation in decomposing this black box into sub-black boxes. Accuracy is another motivation for decomposition, since a queuing network model will be closer to reality than a single queue model representing the same element. Last, the black boxes model identification process may be quicker and simpler, for smaller black boxes, and may have less weird behaviors than for bigger ones.

A black box is a constituent whose content is unknown. You may only know its external interfaces and be able to invoke their operations, and observe the outputs resulting from your invocations. This black-box may (or may not) provide an interface to give some information about its state. It runs in an execution environment whose resources usage may be observed (CPU, RAM, network bandwidth, etc.). We particularly address software black boxes running on an operating system. Commercial, off-the-shelf software elements, as well as complex open source middleware, would be typically black boxes. In case of distributed software, network interactions give decomposition opportunities.

C. Automatic model identification

Once we have decomposed the global system into black boxes, we need to get a performance model for each of them, and then to combine these models into a single one representing the global system. To achieve this, we choose to model black boxes as queues, and the global system as a queuing network. The idea is that we can experimentally identify queuing models that best represent the performance of black boxes, and then build the resulting queuing network for performance prediction. Model identification is based on non-parametric statistical tests. This enables to determine the best distributions fitting service times and inter-arrival times.

The other idea is to get experiments on black boxes automatically performed by a load testing platform, enhanced with self-regulated load injection capabilities [3]. The workload is automatically adjusted according to measures and policies that define workload steps, levels and saturation criteria. There are three reasons for this step-by-step increasing workload injection. First, we have no knowledge and we make no assumption about the maximum capacity of each black box: we start with a minimal capacity assumption, and then we gradually increase the assumed capacity. Second, we prevent load injection from actually reaching a critical saturation level that would result in a black box crash, with a possible necessity to reboot and restart. Third, we want to observe the black box in permanent, stable states, which practically requires to have these steps.

This experimental process uses research results in terms of component-based architecture for building autonomic computing systems [4].

D. Performance prediction

Once the Knowledge part of our autonomic system is fed with the black boxes queuing models, the Analysis function of any self-* control loop is able to evaluate performances of possible target configurations. This prediction may be based on

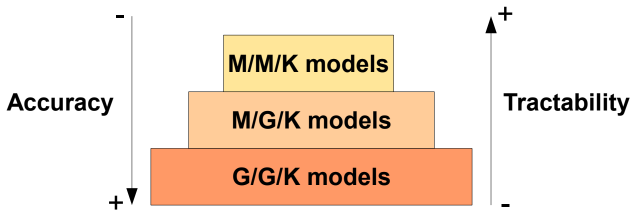

Figure 1. Accuracy versus tractability

queuing network simulation or analysis. When several queuing model candidates have been successfully identified for a single black box, the actual model selection may be driven, on the one hand, by its accuracy, and, on the other hand, by its ability to be quickly analyzed or simulated. The more accurate is the model, the more difficult is the analysis (see Figure 1).

As a matter of fact, the efficiency of this performance prediction influences the speed and effectiveness of the self-* control loop. The global process is summarized by Figure 2. This paper develops the second step (model identification) of the approach, as a continuation of [2]. An example of the use of identified queue models in performance prediction is given.

This paper is organized as follows: first, we position our work with other related work in Section II. Then, Section III describes how the self-regulated load injection process is achieved: we compute the duration of an injection period and explain how to estimate stabilization time, injection step duration, and sampling period. In Section IV, we detail the black box model identification process by first presenting inter-arrival and service sampling, and then by explaining how to determine the distribution shape and the whole identification process. We also present how to estimate, from the observed parallel processing level of a given black box, the corresponding queue model's number of servers. In Section V, we show how to use identified models in performance prediction. We show a practical application of our model identification process in Section VI on a typical use case and we give experimental results. Finally, we conclude in Section VII and give some open questions and perspectives.

II. Related work

Several works have been proposed to model systems for autonomic computing purposes. Some authors used regression models [5] for transactional systems, but most of them proposed queuing networks as predictive models [6], [7], [8], [9]. Kamara et al. [7] modelled a 3-tiers architecture with a single queue; Rafamantanantsoa et al. [8] described a simple web server with an M/G/1/K-PS queue model. The parameters of this model (queue capacity and mean service time) are estimated by the maximum likelihood technique, given data obtained by extensive experiments. Other proposals [9] used queuing networks instead of a single queue model. This last modelling seems to be more appropriate for distributed systems.

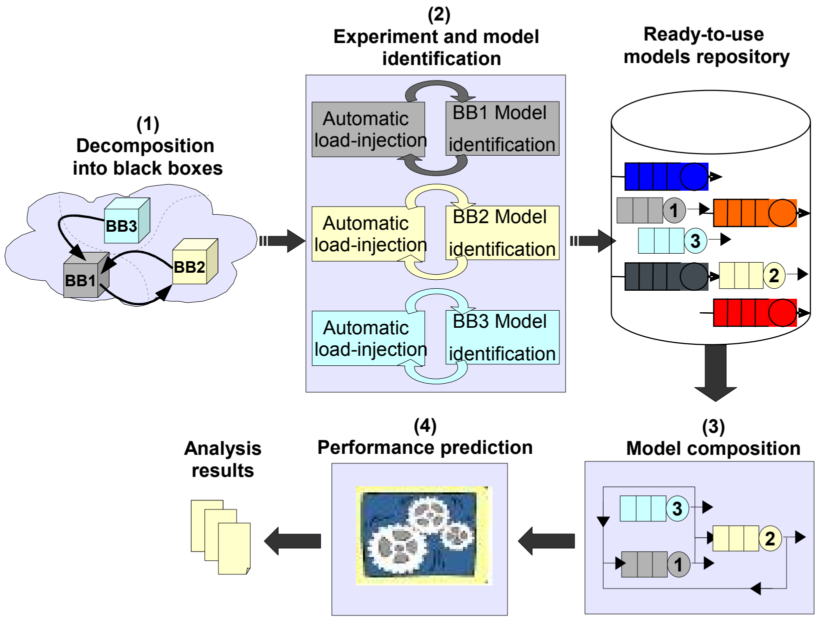

Figure 2. Performance prediction of a distributed application configuration

Begin et al. [10] approximate the measured behavior of a variety of systems by selecting and calibrating a limited set of queuing models. More recently, using a black box approach, Menasce [11] addresses the problem of finding an unknown subset of service demand parameters in queuing network models, given the known values and given the values of response times for all workloads.

Woodside, Zheng and Litoiu [12] worked also on tracking parameters of queuing network models for an autonomic system. They used extended Kalman filters, while integrating various kinds of measured data such as response times and utilization. Using Kalman filters is quite valuable since they are known to be predictor-corrector estimators: they make the obtained model optimal when dealing with error covariance minimization.

These proposals are interesting, however autonomic systems need to be dynamically analyzed with precision, to be able to choose the best solution when a problem occurs.

This fact led us to estimate an accurate queuing model, that might represent the observed system: we propose to model inter-arrival and service times, as well as the number of servers. We don't use for that Kalman filters because they are not suitable for our approach: first, Kalman filters are not sufficient in our case, since we estimate shapes of distributions. Second, convergence of these filters is not guaranteed for a number of queuing models, which makes their use without a predefined model more difficult in an autonomic approach. Rather, we can identify distributions with more precision, using non-parametric statistical tests. Most of our experiments show more Lognormal and other distributions than exponential distributions, which were used in most work.

Our approach is thus a generalization of previous methods proposed in literature. It also provides rich distributions modelling systems behavior, and giving more information. The final contribution of this paper is an implementation of the developed approach in a prototype for modelling multi-tier autonomic systems and anticipating their performances.

III. Self-regulated load injection

Our approach relies on injecting a step by step increasing workload (see Figure 3). To allow estimation of a coherent model, we inject a workload composed of a single traffic type. Basically, this consists in automating a benchmarker work, trying to find the performance limits of a system through load testing. It injects a first load level, observes the system behavior (response time, resource usage…) and decides the amount of the next workload step. It repeats the procedure until reaching - or more probably overpassing - a workload high limit, beyond which the system becomes unstable or the delivered quality of service is no more satisfactory.

To automate this process, we rely on a load injection framework: the CLIF [13] framework provides injectors, for generating a workload modelled as *virtual users* (vUsers) and measuring requests response times, and probes, for measuring usage of arbitrary computing or networking resources. Moreover, we need to define an injection policy specifying several parameters, mainly: the workload level in each step (injection step), the length of an injection period, the time required to get the system in a stable state (stabilization time), the System Under Test saturation limits where the load testing process must stop.

In the remainder of this paper, we use the Kendall notation [14] of an elementary queuing system, denoted by $T/X/K$ where T indicates the distribution of the inter-arrival times, X the service times distribution and K the number

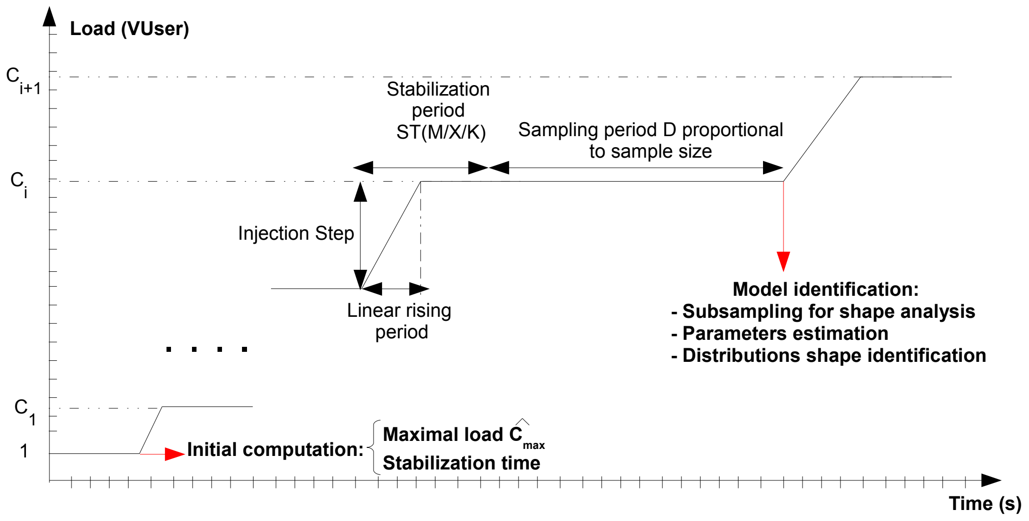

Figure 3. Model identification during a ramp of load injections

of servers ($K \geq 1$). Note the number of servers represents the observed parallel processing capability and not the actual number of physical computers or processors. This capability practically depends on the multi-threading support and on the computation profile (e.g., CPU-intensive or Input/Output-intensive). So, it is both hardware-dependent and software dependent (operating system, middleware, application). As a simplification, we still consider it as an integer number in this, but it is more likely a decimal number.

A. Injection policy

The main issue related to self-regulated injection is to determine automatically the injection policy parameters defining the steps of increasing workload (see Figure 3). These parameters are computed at runtime, step by step.

1) Estimation of maximal load: An initial load injection phase is undertaken to estimate the maximal supported load $\hat{C_{max}}$. In this phase, we load our system with markovian interarrivals requests of one virtual user. We collect response times and compute a first approximation of $\hat{C_{max}}$, as $\frac{1}{\mu}$, μ being the service rate. This result comes from the fact that, when dealing with one customer arriving in an empty queue (no concurrence), the mean waiting time is null ($\overline{W}$=0), leading to the following mean response time:

$$\overline{R} = \overline{W} + \overline{X} = \overline{X} = \frac{1}{\mu}$$

When the queue model is M/G/1, the arrival rate of requests converges to μ. An example of this convergence is depicted in Figure 4, obtained when experimenting our example. The value of $\hat{C_{max}}$ is experimentally corrected when the estimated number of servers K increases (see Section IV-D).

2) Injection step: The load injection step should be carefully defined, as a small step may result in a huge experimental time, whereas a big step may brutally saturate the system. We use an additive increase while checking if the experiment is close to the value of the estimated maximum load $\hat{C_{max}}$. The

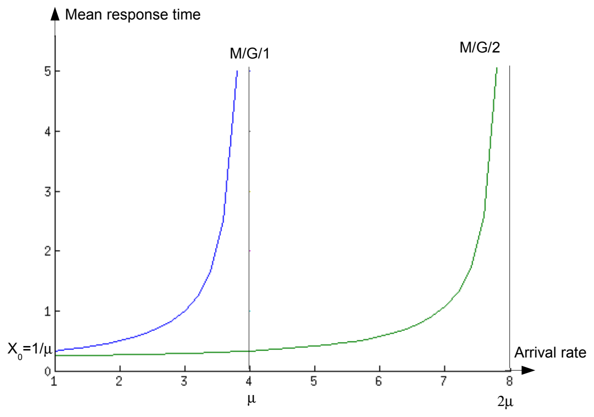

Figure 4. First estimation of maximal load Cmax

increment is defined through a decomposition of the estimated maximum workload into a user-set number of iterations. The greater this parameter is, the more workload steps will be performed, thus giving more accurate information, but taking much more execution time.

3) Rising period: After an injection phase, the load is increased with an increment and submitted. To avoid a malfunctioning of the system due to a big injection step, we choose to inject the increment of requests gradually, drawing thus a ramp. The rising period is the period during which the injection of a new load increment is done. In practice, injecting 10 vusers/sec is acceptable. The rising duration is then automatically computed as a function of the injection step, while maintaining 10 vusers/sec.

4) Sampling period: This should be computed such that the system behavior remains stationary and the sampling is sufficiently large to get good confidence in measures. Rai Jain in [15] proposes a formula for determining the sample size n required to achieve a given level of accuracy $r\%$ and a confidence confidence interval of $100 * (1 - \alpha)\%$:

$$n = (\frac{100 z \sigma_n}{r \bar{m}})^2,$$

where z is the normal variate of the desired confidence level (for a 95% confidence interval, $z \approx 1.96$), $\bar{m}$ is the mean value of the parameter to estimate and σ_n stands for the sample standard deviation.

B. Estimation of the stabilization time

When collecting measures, it is important to distinguish the transient and stationary periods. The variance of measured data gives a first insight in system stability. This is not sufficient as there may be measurements peaks when some phenomenon like the garbage collector appears. Thus, a combination of theoretical and experimental methods is required.

We estimate the stabilization time at each injection step, as the convergence time of the Markov chain [14], [16] underlying the associated queue model. We restrict ourselves to Engset models.

In other words, the stabilization time ST is considered as the time required to get the equilibrium (stationary) state probabilities, denoted as the probability vector π, when the Markov chain is ergodic.

It can be computed by studying the transient behavior of the system. As the queue model of the step (i) is not yet defined, we rely on the queue model (denoted $model_{(i-1)}$) determined in the previous step (i-1). We compute ST by:

(1) deriving the transition probability matrix P of $model_{(i-1)}$, which has a dimension equal to MxM, M being the amount of load submitted in step (i);

(2) obtaining the probability vector:

$$\pi^{(n)} = \pi^{(n-1)} P = \pi^{(n-2)} P^2 = \ldots = \pi^{(0) P^n},$$

$\pi^{(0)}$ being the initial vector and n the number of iterations required to reach the equilibrium state;

(3) computing the stabilization time ST as:

$$ST = \frac{n}{\lambda + \mu},$$

λ being the inter-arrival parameter of step (i) and μ is approximated by the service rate parameter of $model_{(i-1)}$.

As we base on $model_{(i-1)}$, which is not necessarily the model of step (i), we correct the obtained stabilization time by adding an error ϵ_i, computed experimentally by observing the variation coefficient of measures collected in step (i).

IV. Identification of the performance model of a black box

As previously said, a black box is modeled with a queuing model. Identification of such model requires to define the distribution of inter-arrival times, the distribution of service times and the number K of servers. The identification of these distributions requires first to capture adequate measures from the injection framework, then deduce a sample to analyze, i.e., an interarrival sample and a service sample. The obtained samples are submitted to statistical tests from which is estimated the corresponding distribution shape. Parameters of the identified distributions (interarrival and service) are also estimated to complete the definition of final models.

A. Inter-arrival sampling

This implies, for a distributed system, to identify the shape of the inter-arrival process received on upstream of each black box. The interarrival process of a black box depends, on one side, on the rate of submitting requests to the global system and, on the other side, on the system architecture. As a matter of fact, we investigate inter-arrival times distribution of a black box only when being in the context of a configuration of the system to which it belongs. To achieve that, load injections are submitted to the system and arrival times of requests are captured for each black box. For a given black box, we compute inter-arrival times, obtaining the sample T.

B. Service identification process

This process consists in submitting load injection to the black box (see Figure 3). The workload is increased through several steps, until reaching the maximal estimated load $\hat{C_{max}}$. As many theoretical results exist for the M/G/1 and M/G/K models, we choose to inject requests through exponential inter-arrival times, obtaining at worst a M/G/K model.

Let us detail this process. It is done through two major phases : an initialization phase and an identification phase.

1) Load injection steps: Two major steps are followed:

a) Phase 1: Initialization: This phase consists in submitting to the black box a flow of similar requests representing only one customer. We measure the response time mean, denoted $\overline{R_0}$, during the sampling period computed as explained in Section III-A4. As a single customer uses only one server, we can infer that the black box behavior follows the M/G/1 model in the worst case (see Section III-A1). Hence, the value of service rate during this phase, μ_0, is used to get the maximal estimated load C_{max} and the next injection step.

Note that a realistic traffic typically involves a mix of different kinds of requests, e.g. user connection and authentication, requests involving or not database read or write operations, etc. In fact, we could also address such heterogeneous traffics, as long as response times are of the same order of magnitude from one request kind to another. Our steps might last longer because of the higher service time variability, but the resulting average service time would be still representative of the traffic mix. We would assume that the mix is the same whatever the workload level. Experiments about this would be an interesting complement to our work.

b) Phase 2: Identification: This phase is carried out through several steps , where each step (i) consists of:

1) Submitting a self-regulated load injection C_i following a Poisson distribution.
2) Waiting for stabilization (stabilization time already computed as shown in Section III-B) and collecting experimental measures during the computed sampling period: response times, interarrival times and utilization of the black box for this workload step.

3) Inferring service times $(X_k)_{1\leq k\leq n}$ from the samples of response times $(R_k)_{1\leq k\leq n}$ and interarrival times $(t_k)_{1\leq k\leq n}$.
4) Removing aberrant values from the service time samples. This is done by removing a fixed percentage (for instance 5%) of greater values. These great values may be considered as experimental measurement errors, resulting, for instance, from by some phenomena such as the occurrence of garbage collector on the load injector.
5) Identifying the shape of the service times sample using statistical tests.
6) Computing the injection parameters for the next step: injection step and the stabilization time.

2) Stop condition based on saturation checking: During the load injection steps, it is necessary to test if the black box is getting saturated to stop the experiment. This is done by monitoring the black box state and detecting whether its utilization reaches some predefined limits. In our context, we define the black box utilization through computing resources utilization (CPU, memory, JVM heap memory). This is achieved by deploying a probe for each monitored resource. Load injection is stopped as soon as one or several resources get(s) saturated. Resource saturation is defined as reaching a given threshold, determined by an expert of the system.

When the black box reaches the estimated maximal load and no saturation appears, we correct the maximal load and continue load injection tests, and so on, until saturating the black box (see Section IV-D). This technique of reaching maximal load level allows us to capture all possible behavior of the box against all possible load levels. Hence, the obtained model is the closest and the best one fitting the service offered by the black box per load level.

3) Service sampling: To obtain the service sample of an injection step, the load injection framework delivers several measures. We use mainly response times $(R_k)_{1\leq k\leq n}$, interarrivals $(t_k)_{1\leq k\leq n}$ and utilization U of all resources. We need also to estimate the scheduling policy to be able to compute the service sample. So, in a first time, we assume that the black box relies on a process sharing (PS) scheduling policy, then when getting close to saturation, the scheduling policy becomes FIFO. In both cases, service times $(X_k)_{1\leq k\leq n}$ are computed as follows:

1) For a PS policy:
$$X_k = R_k * (1 - \lambda * \overline{X}) \text{ [14]}$$
where λ is the interarrival rate used during the load injection and $\overline{X}$ is estimated with the fix point algorithm using the estimated $\overline{X}$ of the previous step as an initial value.
2) For a FIFO policy: We use an extended result relating service times $(X_k)_{1\leq k\leq n}$, response times $(R_k)_{1\leq k\leq n}$ and interarrival times $(t_k)_{1\leq k\leq n}$ [14]:
$$R_k = [R_{k-1} - t_k]^+ + X_k$$
This result is valid for a model using one server and a FIFO policy. So, if we get to identify, in step (i), a model characterized by K servers (K>1), this result cannot be used. To generalize this result, we propose to use a similar result:
$$R_k = [R_j - t_{j,k}]^+ + X_k$$
where j corresponds to the previous request that quitted the server, which served the k^{th} request, and $t_{j,k}$ is the interarrival between the j^{th} and k^{th} requests. The j^{th} request is determined by recomputing iteratively service times $(X_k)_{1\leq k\leq n}$, beginning from the first served request and using the R_j and $t_{j,k}$ computed from collected measures.

C. Distribution shape identification

To automatically determine the shape of inter-arrival times and service times distributions, we use a statistical test based approach, which selects the distribution that fits well the samples.

1) Identification using statistical tests: The statistical goodness-of-fit hypothesis test is a process that consists of making statistical decisions using experimental data. Several hypothesis testing approaches exist. In our case, we use the Kolmogorov-Smirnov statistical test [17], since it is appropriate to continuous distributions. We also use the Anderson-Darling test, which is appropriate to distributions with heavy queue.

However, these tests are only suitable for small samples and cannot apply to large samples. To avoid this drawback, we uniformly select a sub-sample from our data, on which we perform the test. Distributions that give a p-value (output value of a statistical test) greater than 0.1 are selected as good distribution representatives for our measures.

2) Estimating distributions shape: To seek the most appropriate distribution fitting an inter-arrival/service sample, we test several distribution families, known in the literature as distributions appearing naturally in computing systems [18]: exponential family, heavy-tail distribution family, etc.

We begin by choosing a distribution from the exponential family. We estimate parameters of each distribution using a *Maximum likelihood* estimator. We then keep distributions that give a p-value greater than 0.1.

To achieve that, we compute the variation coefficient CV^2 of the sample and its confidence interval. Depending on its value, we test a set of distributions. If the confidence interval of CV^2 contains 1, we test the exponential distribution. If $CV^2 \in]0, 1[$, we test the hypoexponential(k) distribution, the Erlang(k) distribution and the gamma distribution. If $CV^2 \in]1, +\infty[$, we test the hyperexponential(k) distribution, the Uniform, the Normal, Lognormal, and Weibull distributions.

For each distribution d, we analyze a sample S as follows:

- If necessary, we make transformations (for instance a shift) on the sample $(S_k)_{1\leq k\leq n}$ to fit distribution d,
- We estimate parameters of d with a *Maximum likelihood* estimator.
- We choose a small sample S^* from $(S_k)_{1\leq k\leq n}$.
- We perform a statistical test for S^* and d with estimated parameters. As previously said, we choose to work with the Kolmogorov-Smirnov test. We repeat several times

this statistical test, and take the mean of obtained p-values, so as to ensure a correct p-value result.

- We then discard distributions whose statistical test gives a p-value less than 0.1. The set of remaining distributions, denoted L, is considered as the possible behavior of the black box service, resulting in a black box model M/X/K specified by several service distributions.

D. Estimating the number of servers

The number of servers (parallel processing capability, see section III) observed for a black box is determined experimentally. When reaching the estimated maximal load $\hat{C_{max}}$, we observe the black box utilization. If it indicates the black box is saturated, the number of servers remains 1. Otherwise, we progressively (step-by step) increase the load and check if saturation is reached. If no, we correct the estimated maximal load $C_{max} = k * \hat{C_{max}}$ and increment the assumed number of servers by 1. We resubmit new increasing load injections. We observe again the black box utilization and repeat the procedure until reaching saturation.

If during step (i), we identify a number of servers K>1, we need to correct models of previous steps, so we repeat samples analysis of these previous steps, by recomputing the service sample and re-identifying the models of each step.

Note that the estimation of servers number representing a black box is done independently from distributions shape estimation. The final estimated number is deduced at the end of the step-by-step process (at saturation), while shape distributions are estimated at each injection step.

E. Validation of the black box queue model

The identification process produces one or several candidate queue models possibly corresponding to different load levels. These models are validated by comparing empirical performance measures with theoretical ones, typically mean response time, mean waiting time and throughput.

V. USING IDENTIFIED MODELS IN PERFORMANCE PREDICTION OF AUTONOMIC SYSTEMS

Let us consider a system configuration C as a possible solution for ensuring an autonomic feature. The goal is to be able to evaluate performances of C before its application on the system.

To reach this objective, the first step is to feed the autonomic system with its black boxes queuing models, following the identification process sketched in Section IV: a model repository for the system is hence created. Then, the global model of the configuration C is built, so that to launch the Analysis function of any self-* control loop and predict performances of C. This is done by picking from the model repository and by composing them.

A. Composition of black box models

A queuing network is entirely defined by the number of its nodes, the parameters of each node (queue) and the routing probabilities between nodes (probability that a request is transferred to the j^{th} node after service completion at the i^{th} node). To compose the set of black boxes models, it is important, in one side, to get the topological structure of the interconnection, and in the other side, to describe transitions between the models in this topology.

1) Transitions between black boxes models: To compute the routing probabilities between nodes, we rely on traces of incoming and outgoing traffic of each node. We propose so to conduct a typical experimentation, during which we capture input and output requests of each black box. The capture is done using log files and is specific to each software product. The number of outgoing requests of each black box is deduced from this capture and so are the incoming requests to the corresponding black box addressees (notice that common log files give generally for each incoming request the sender address). A ratio of the traffic distribution between the black boxes is then computed, resulting in the definition of routing probabilities.

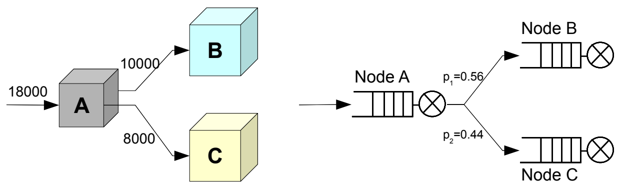

Figure 5. A black box interconnection example

Let us take an example of a system S (Figure 5), made of three black boxes A, B and C, where A is linked to B and C and each of B and C are only linked to A. In this case, we compute the number of outgoing requests of A and the number of incoming requests of each of B and C. Let us say there are 18000 outgoing requests for A, 10000 incoming requests for B and 8000 incoming requests for C. The routing probabilities are $p_1 = 10000/18000 = 0.56$ between A and B and $p_2 = 8000/18000 = 0.44$ between A and C.

2) Building the global model of a system: Once the transition probabilities between black boxes queues are obtained, we compose the queues and get a queuing network, the global model of the system. In our models, we deal only with open networks, as we study distributed infrastructures where requests are received and leave the system after service processing completion.

B. Analysis of the global model

To predict performances of the configuration C, we solve the obtained queuing network model using a specific algorithm allowing the computation of theoretical performance parameters. Typical parameters are:
(i) for the whole system: mean response time $\overline{R}$, throughput D and mean customer number $\overline{N}$;
(ii) for each queue Q_i: mean response time $\overline{R_i}$, mean number of customers $\overline{N_i}$ and utilization U_i.

)

The resolution algorithm to use depends on the complexity of queues composing the whole model :

- If the queuing network is composed of only M/M/K models, the exact MVA (Mean Value Analysis) algorithm is the most suitable to use [14], [19]. This algorithm is suitable for many systems as the markovian distributions are known in the literature to appear naturally in various systems [18].
  The MVA method allows to compute the mean values of parameters of interest such as the mean waiting time, throughput and the mean customer number at each node. Another algorithm to use is the AMVA algorithm [14], [19], which is an approximation improving the computation time of MVA.
- In other cases and depending on the structure of the resulting queuing network, we use the appropriate algorithm such as the method of Raymond Marie [20], [21]. This algorithm has been defined as an approximate solution for studying the asymptotic behavior of a network of queues with a general service distribution. When the network is composed of different types of queues, we propose to compute performance bounds. In the worst case, when analysis is impossible, we use simulation to determine global performances.

VI. Illustration

The automatic model identification process is currently implemented as a framework prototype. This framework provides: (i) an automated benchmarking controller, based on CLIF [13], for performing the self-regulated load injection steps, (ii) a model identification tool, based on Matlab/R statistical environments [22], [23] and, (iii) an editor for composing identified queue models and launching analysis/simulation of obtained queuing networks for performance prediction.

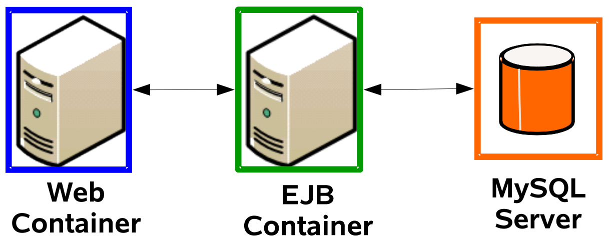

Figure 6. Use case: SampleCluster JONAS application

To illustrate the steps of our identification process, we experimented a three-tiers Java EE application, called *SampleCluster*, that runs in the *JONAS* application server, an open source Java EE implementation developed by the OW2 consortium [24]. This application was developed as a testing application of a JONAS cluster. This cluster is composed of a Tomcat web container server, an Enterprise Java Beans (EJB 3.0) container and a MySQL database storing EJB sessions information (see Figure 6).

The system is decomposed into three black boxes: the Web container tier, the EJB container tier and the database tier. This first level decomposition matches exactly the multi-tier architecture and this is very useful to operate an adequate and good system sizing. These black boxes are modeled using our automatic identification process. To inject requests to a black box, we use CLIF load injectors. We use a load injection scenario featuring virtual users whose behaviors represent real user behaviors. Network latency is considered as negligible for these experiments, since we operate with a high-speed local area network (Gigabit Ethernet), whose latency order of magnitude is microseconds. To get resources utilization measures and detect system saturation, CLIF probes have been used for CPU, JVM heap memory and RAM. We defined the black box saturation limits as 80% for the CPU (high limit), 80% for RAM (high limit) and 5% for the JVM's free heap memory (low limit).

To model a black box, it must be isolated from the other black boxes. Isolating the database tier is straightforward, since it is the last tier and it does not call any other black box. Isolating the Web and EJB tiers requires more work:

- either develop respectively an RMI plug and an SQL plug, i.e., a fake RMI or SQL server that accepts requests and give responses in place of the real server, in deterministic response time that can be subtracted from the black box measured response times. This technique requires some non-trivial programming, since responses must hold correct information. A record-and-replay solution may be applied, by observing requests and responses with the real server and then replaying known responses on known requests with the plug. This is not too complex to achieve with text-based protocols (such as SQL) when no socket secure layer is used (cf. encryption), but it is much more complex with binary protocols like RMI. Moreover, plugs must be benchmarked in order to know its response time and to be able to compute the black box service times;
- or follow a step-by-step approach, starting the modeling process from the downstream black box (the database black box in our example). In next step, we run the modeling process on the previous black box (EJB container) and, thanks to the model obtained during the first step, we subtract response times of the last box to the measured response times in order to be able to compute the corresponding real service times. Then, we iterate for next steps until the first tier is reached.

We used the second solution (step-by-step) for practical reasons: not only do we avoid developing a plug, but we also avoid benchmarking the plug, while in the step-by-step solution, next black boxes are benchmarked de facto.

In the following, we present modelling results for the three black boxes, then we give performance prediction of the system and an example of fulfilling an autonomic feature.

A. Modeling the database black box

The database black box runs on a Linux server with two 1.4 GHz PIII processors, and 1 GByte of RAM. We used a Linux

Load	Identified Model(s)	Parameters
1	$M/\Gamma/4$, **M/LN/4**, M/Norm/4, M/Wbl/4	m=-8.010, sigma=0.089
6	$M/\Gamma/4$, **M/LN/4**, M/Norm/4, M/Wbl/4	m=-8.034, sigma=0.276
11	$M/\Gamma/4$, **M/LN/4**, M/Norm/4, M/Wbl/4	m=-8.051, sigma=0.333
16	$M/\Gamma/4$, **M/LN/4**, M/Norm/4, M/Wbl/4	m=-8.020, sigma=0.396
21	$M/\Gamma/4$, **M/LN/4**, M/Norm/4, M/Wbl/4	m=-8.030, sigma=0.421
26	$M/Hr_2/4$, $M/\Gamma/4$, **M/LN/4**, M/Norm/4, M/Wbl/4	m=-8.053, sigma=0.428
31	$M/Hr_2/4$, $M/\Gamma/4$, **M/LN/4**, M/Norm/4, M/Wbl/4	m=-8.033, sigma=0.464
36	$M/Hr_2/4$, $M/\Gamma/4$, **M/LN/4**, M/Norm/4, M/Wbl/4	m=-8.074, sigma=0.476
45	M/M/4, $M/Hr_2/4$	p=0.055,$mu_1 = 0.945$,$mu_2 = 465.9$
54	M/M/4, $M/Hr_2/4$	p=0.036,$mu_1 = 0.964$,$mu_2 = 239.3$
63	M/M/4, $M/Hr_2/4$	p=0.064,$mu_1 = 0.936$,$mu_2 = 453.0$
72	M/M/4, $M/Hr_2/4$	p=0.060,$mu_1 = 0.940$,$mu_2 = 419.5$
86	M/M/4, $M/Hr_2/4$	p=0.062,$mu_1 = 0.938$,$mu_2 = 452.0$
100	M/M/4, $M/Hr_2/4$	p=0.065,$mu_1 = 0.935$,$mu_2 = 510.8$
119	M/M/4, $M/Hr_2/4$	p=0.040,$mu_1 = 0.960$,$mu_2 = 231.0$

Table I
MODEL IDENTIFICATION RESULTS FOR THE DATABASE BLACK BOX

server with two 2.8 GHz Xeon processors and 2 GBytes of RAM as a load injector. The automated self-regulated load injection phase was carried out within 14 workload steps, reaching more than 120 virtual users in 5 minutes on average. The saturated resource was the CPU with a 82% usage.

The model identification tool got measures and computed the corresponding service time samples for each workload step. Each service time sample was analyzed using goodness-of-fit tests and graphical methods. Various distributions were identified with their parameters, including the Exponential, Hyper-exponential with two stages, Log-normal, Gamma and Weibull. The candidate models identified during experimentation are given in Table I. Notations for this table I are the following: λ_i refers to the interarrival rate, μ_i the service rate and $\overline{X}_i$ its mean service time. LN refers to the Lognormal distribution, Hr_2 to the Hyperexponential with two stages and Wbl to the Weibull distribution.For each load level, we select the most appropriate model (given in bold characters) according to the statistical tests best results (best p-values and fitting scores). We also give the best model's parameters (*Parameters* column): λ is the inter-arrival rate, μ the service rate for the exponential distribution (μ_1 and μ_2 for the Hyperexponential distribution and p is its probability to go to a stage), μ, σ are the shape and scale parameters of the Lognormal and Normal distributions, and a,b are the Γ distribution parameters.

As the table shows, for light and medium loads (load levels varying from 1 to 36 virtual users), we select the M/LN/4 model, as the statistical tests gives greater p-values for the lognormal distribution. For higher loads, we select the $M/Hr_2/4$ model. Graphs of Figure 7 show service times histograms with identified fitting distributions.

B. Modeling the EJB container black box

The EJB tier runs on a Linux server with two 2 GHz Xeon processors, and 1 GByte of RAM. We used a Linux server with two 2.8 Ghz Xeon processors, and 2 GBytes of RAM as an injector machine. The automated self-regulated load injection phase was carried out within 12 injection steps, reaching more than 162 virtual users in 8 minutes and 0.2 seconds. Figure 8 shows the resulting load profile. The saturated resource was the CPU with 86% usage.

The candidate models identified during experimentation are given in table II. As the table shows, for light and medium load (load levels varying from 1 to 118 virtual users), we select the M/LN/3 model except for one load level ($M/\Gamma/3$ model for load=55 virtual users). For higher loads, we select the $M/Hr_2/3$ model. Graphs of Figure 9 show service times histograms with identified fitting distributions.

C. Modeling the Web container black box

The Web tier runs on a Linux server with two 1.2 GHz PIII processors and 1 GByte of RAM. We used a Linux server with two 2.8 GHz Xeon processors and 2 GBytes of RAM as an injector machine. The automated self-regulated load injection phase was carried out within 6 workload steps, reaching more than 16 virtual users in 7 minutes and 48 seconds. The saturated resource was the JVM heap memory with 4% free space.

The candidate models identified during experimentation are given in table III. As the table shows, the load levels exhibit either an M/LN/1 or an $M/\Gamma/3$ model. Graphs of Figure 10 show service times histograms with identified fitting distributions.

D. Validation of the obtained global model

Our validation of the identified models is two-fold:

- First, we build a global queuing network model representing the SampleCluster application, using the three black boxes' models, and we perform its performance analysis or simulation at a given load level (16.2 requests/s). Then, we compare obtained performance values with real measures at the same workload level, to check the accuracy of our modelling.
- Second, we apply the automated load injection and model identification process on the whole SampleCluster architecture considered as a single black box. Then, we do

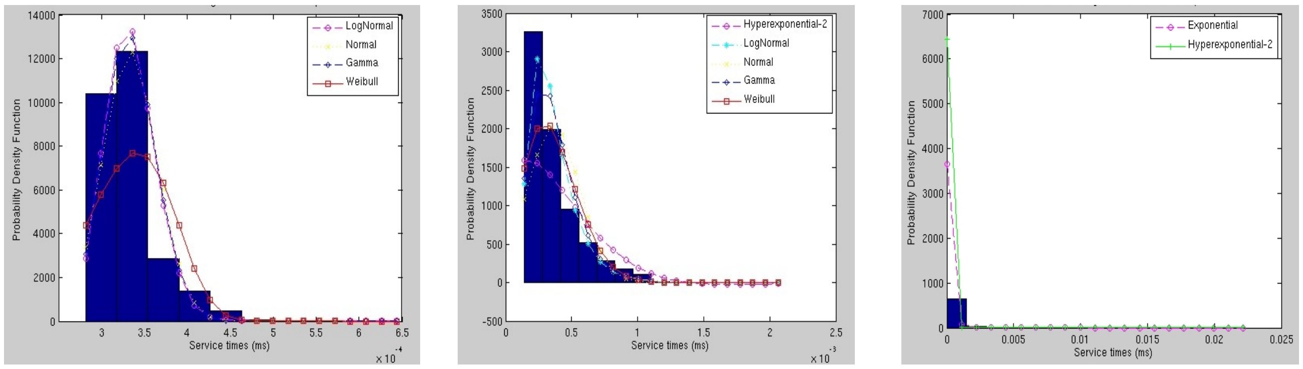

Figure 7. Service sample analysis for the database black box: light (left), medium (middle) and heavy (right) loads

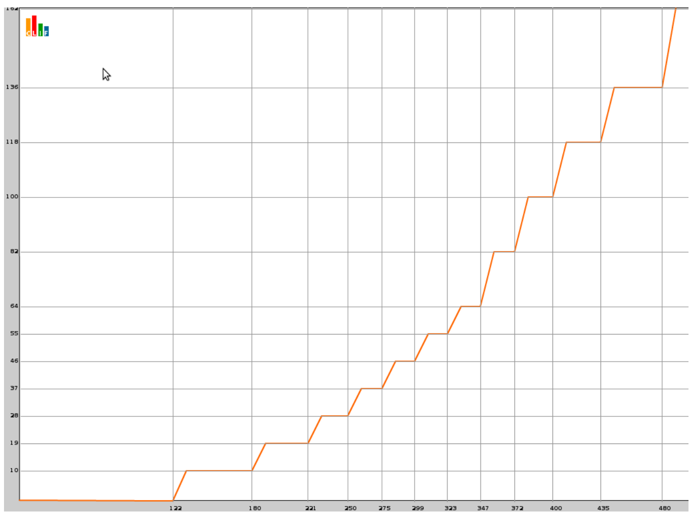

Figure 8. Load profile resulting from self-regulated load injection performed on the EJB tier. The X axis is time, from 0 to 480 seconds. The Y axis is the number of active virtual users, from 1 to 162. 12 steps have been completed, and the 13th step has been aborted because of system saturation detection.

Load	Identified Model(s)	Parameters
1	$M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.282, sigma=0.132
10	$M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.458, sigma=0.166
19	$M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.545, sigma=0.202
28	$M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.602, sigma=0.189
37	$M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.653, sigma=0.247
46	$M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.708, sigma=0.263
55	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	a=15.828, b=0.001
64	$M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.708, 0.306, sigma=0.476
82	$M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.739, sigma=0.363
100	$M/Hr_2/3$, $M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.818, sigma=0.376
118	$M/Hr_2/3$, $M/\Gamma/3$, **M/LN/3**, M/Norm/3, M/Wbl/3	m=-4.792, sigma=0.381
136	M/M/3, $M/Hr_2/3$, M/Norm/3	p=0.138, $mu_1 = 0.862$, $mu_2 = 77.65$
162	M/M/3, $M/Hr_2/3$	p=0.094, $mu_1 = 0.906$, $mu_2 = 27.25$

Table II
Model identification results for the EJB container black box

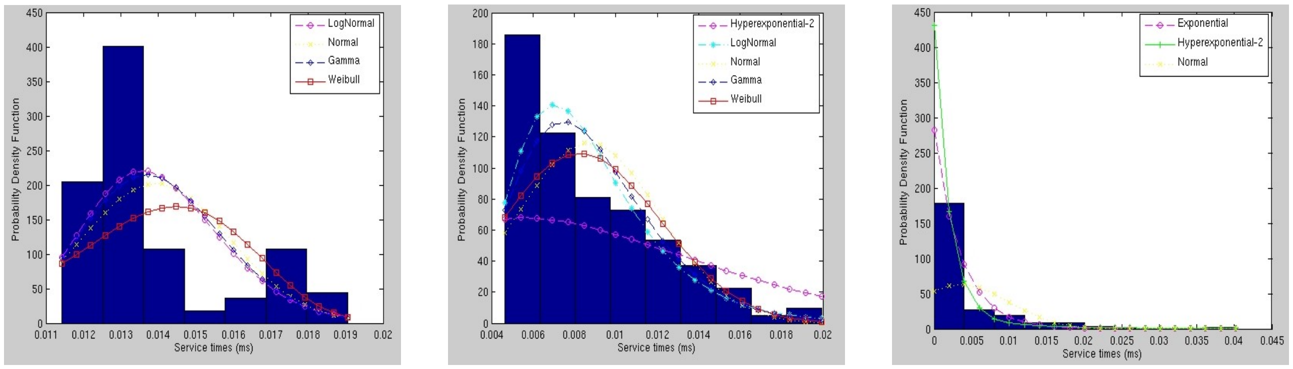

Figure 9. Service sample analysis for the EJB black box: light (left), medium (middle) and heavy (right) loads

Load	Identified Model(s)	Parameters
1	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	a=76.87, b=0.001
4	$M/\Gamma/1$, **M/LN/1**, M/Norm/1, M/Wbl/1	m=-3.254, sigma=0.133
7	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	a=62.92, b=0.001
10	$M/\Gamma/1$, **M/LN/1**, M/Norm/1, M/Wbl/1	m=-3.493, sigma=0.145
13	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	a=36.82, b=0.001
16	$M/\Gamma/1$, **M/LN/1**, M/Norm/1, M/Wbl/1	m=-3.679, sigma=0.166

Table III
MODEL IDENTIFICATION RESULTS FOR THE WEB CONTAINER BLACK BOX

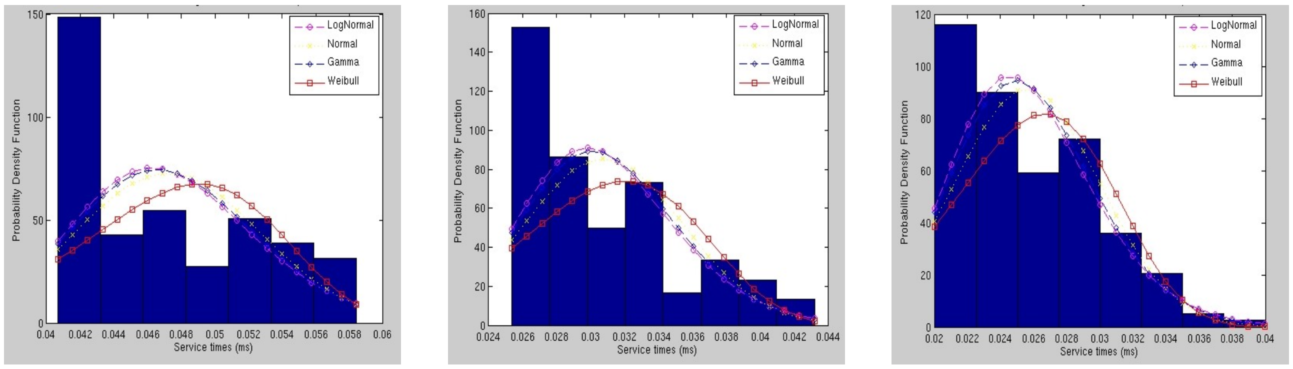

Figure 10. Service sample analysis for the WEB black box: light (left), medium (middle) and heavy (right) loads

Performance index	Estimated value for the system decomposed as 3 black boxes	Estimated value for the system seen as a single black box	Measure
Response time	52 ms	51 ms	52 ms
Throughput	16.2 requests/s	16.2 requests/s	16.2 requests/s
Clients number	0.87	0.80	-

Table IV
COMPARISON BETWEEN THEORETICAL AND EMPIRICAL PERFORMANCE INDEXES

performance analysis of the obtained model at the same load level and we compare with the queuing network results and the real measures. The goal here is to check the model accuracy, and especially to see if decomposing the whole system into three black boxes gives more accurate results than when considering a single model for the whole system.

Table IV shows mean values of theoretical performance indexes computed using the identified models of each tier and of the system modelled as a single black box, at the load level of 16.2 requests/s. We see that these values are very close to the mean empirical values. The relative error for the mean response time is 0.75% for the 3-tiers decomposition and 2.47% for the single global model. This is a partial validation of our full automated benchmark and process, on this particular sample application. This result also shows that accuracy is actually better with the 3-tiers decomposition, and with a finer granularity (in our example, relative error is 3.3 times as small), which partially validates the interest of decomposing the global system into several black boxes and building a queuing network.

E. Performance prediction for self-sizing feature

In this section, we sketch an example of autonomic reaction to possible bottlenecks, which may appear in the SampleCluster system. Whenever a bottleneck appears, we show how the Analysis function of the self-sizing control loop is able to find the best system configuration to apply, through performance analysis/simulation of possible target configurations.

Of course, the notion of "best configuration" is a matter of viewpoint. From the system user's viewpoint, only quality of experience criteria count, such as end-to-end response time, service availability and reliability. From the system operator's viewpoint, a trade-off must be found between investment and operating expenditures on the one hand, and client satisfaction on the other hand. Request throughput capability is a good criteria for the operator since it rules how many clients may be simultaneously served by the system. Other criteria such as usage of processor, memory or network bandwidth are also of interest to optimally size the system's resources. However, the operator must also take quality of experience criteria (e.g. end-to-end response time) into account. The best configuration typically consists in minimizing the infrastructure costs, while meeting a service level agreement with respect to given workload assumptions (number of users and resulting workload).

In our example, we assume that a load rate of 180 requests/s is submitted to the system. Performance simulation of current configuration gives a utilization equal to 1 for the Web container black box, thus showing saturation of this tier, and $9778s$ global response time, i.e., 2.71 hours, which is an unacceptable quality of experience.

In this case, the self-sizing control loop would launch a decision process, which chooses the best solution. Possible target configurations are depicted in Figure 11. Table V shows, in one hand, global response time and global throughput for the multi-tier system, and in the other hand, utilization indexes of each tier. These performance results are computed by the performance analysis/simulation function of the control loop. We can see that solution 3 results in an improved global response time and an enhanced throughput. This configuration is hence the best one and more adequate to our multi-tier system, and then will be chosen by the autonomic control loop to be applied to the system.

VII. Conclusion and Future Work

This paper addresses automated performance modelling of software elements considered as black boxes. Our goal is to be able to predict the performance of a distributed application configuration composed of these black boxes, and to use it in autonomic systems so that self-* features can integrate performance awareness while they plan system reconfigurations. Target applications are those being able to evolve to more strengthened configurations, through replication of constituents.

For this purpose, we have proposed a performance model identification process for black boxes. The process automatically delivers, for each black box under test, one or several queuing models with their parameters, according to a number of workload ranges. This process has been implemented as a framework prototype, reusing the CLIF open source load testing platform for workload generation and resource utilization monitoring. The process usability has been assessed through the experimentation of a three-tiers web application and the first results are promising. A first, partial validation is shown on an clustered Java EE sample application, showing a good level of response time prediction accuracy, and even better when the application is decomposed into black boxes instead of considering it as a single black box.

However, some issues and difficulties were met:

- Isolating a black box from dependent servers (i.e., servers that are subsequently invoked by the black box when it processes a request) is mandatory but not straightforward to achieve. Two solutions may be adopted:
  (i) build plugs to replace dependent servers and characterize their performance; this solution is specific to each protocol, and involves programming, benchmarking and possibly some network-level wire-tapping efforts;
  (ii) follow a step by step approach starting from the final black box in the architecture and using identified models of the characterized tier. We preferred the latter solution for it is simpler to implement. But, while the plug can be designed for high performance (it is a fake server), a real server may saturate before the tested black box. In this case, the black box modelling will partially complete, with missing high load steps, because of the bottleneck. The solution is to replicate the dependent server causing the bottleneck.
- We provide no particular support for capturing traffic routing between black boxes. First, we consider a simplified vision of the traffic, assuming a pipe call topology between black boxes, with no feedback calls.

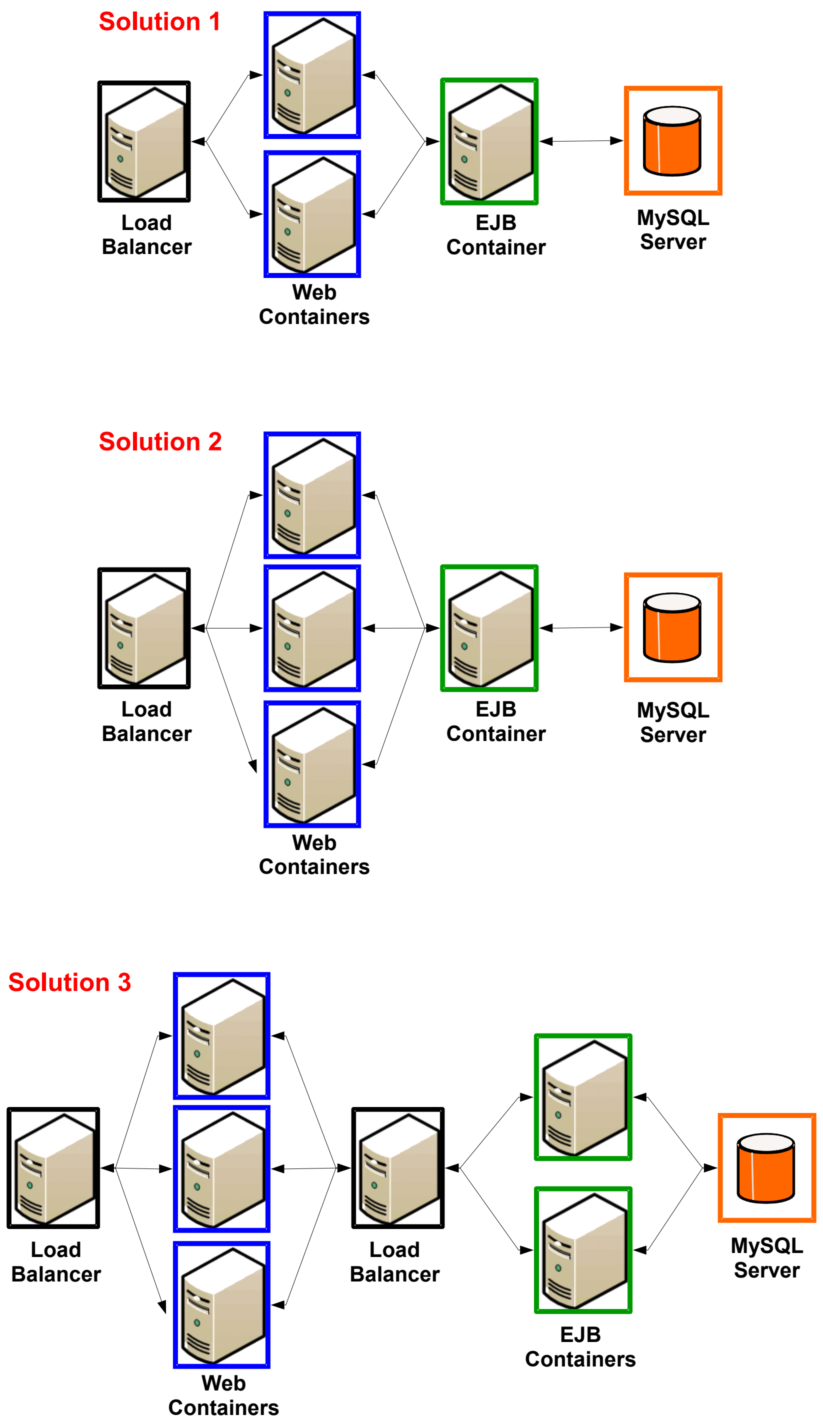

Figure 11. Possible target configuration for solving the detected bottleneck

Performance index	Solution 1	Solution 2	Solution 3
Response time	4186 sec	1855 ms	1454 ms
Throughput	70.88 requests/s	106.80 requests/s	117.20 requests/s
Utilization Web 1	0.90	1	1
Utilization Web 2	0.90	1	1
Utilization Web 3	-	1	1
Utilization EJB 1	0.70	1	0.59
Utilization EJB 2	-	-	0.60
Utilization MySQL	0.09	0.14	0.15

Table V
PERFORMANCE ANALYSIS RESULTS FOR POSSIBLE TARGET CONFIGURATIONS

However, this assumption is met with common multi-tier applications, which are our key targets. Second, we don't provide a solution to capture multiple round-trip calls between black boxes, in which an incoming call in tier n may result in more than a single call to tier $n+1$. However, our queuing network builder supports a multiplication factor, which makes it possible to specify that a given request on black box n generates r requests on black box $n+1$.

- Our work considers a traffic of homogeneous requests. Considering heterogeneous traffics with different request kinds coming with highly variable service times would require some more work. The issue is quite wide if you consider also heterogeneous admission policies depending on the request kind (priorities, preemption, etc.). But this would be typically not the case for the class of multi-tier applications we consider. Complementary experiments would give valuable feedback about the influence of requests heterogeneity in terms of service times on the different stages of the process and the accuracy of final performance predictions.

This work makes little assumptions about observation capabilities of black boxes: response time measurement as it is experienced by a client, and monitoring utilisation of host operating system resources. To improve and extend our framework, it would take some more intrusive observation capabilities. For instance, calls profiling and network analyzer tools should be integrated to help capture information about call routing or to help build plugs.

This work is essentially processor-centric, but it could be extended also for modelling other resources utilization (e.g., network bandwidth, RAM, disk space or disk transfer rate). Similar statistical techniques may also apply, but the set of relevant candidate distributions are likely to differ. This would be valuable for sizing each server, and not only the replication level of tiers.

Finally, our future work concentrates on the autonomic vision, since we plan to integrate this performance prediction platform to an autonomic system manager, responsible for checking or proposing new system configurations matching given performance requirements. Within the SelfXL project [25], applications of this "performance oracle" are foreseen for anticipating and dynamically adjusting the number of virtual machines required for a given service in a cloud computing environment.

REFERENCES

[1] IBM, "An architectural blueprint for autonomic computing," http://www-03.ibm.com/autonomic/pdfs/AC Blueprint White Paper V7.pdf. Last accessed: January 2011, June 2005.

[2] A. Harbaoui, B. Dillenseger, and J. Vincent, "Performance characterization of black boxes with self-controlled load injection for simulation-based sizing," in *Proceedings of CFSE'2008*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 172–182.

[3] A. Harbaoui, N. Salmi, B. Dillenseger, and J. Vincent, "Introducing queuing network-based performance awareness in autonomic systems," in *Proceedings of ICAS'2010*. Cancun, Mexico: IEEE Computer Society, march 2010, pp. 7–12.

[4] ANR Selfware project, "Selfware: Lessons learned to build autonomic systems," http://sardes.inrialpes.fr/~boyer/selfware/documents/SP1-L3-Architecture.pdf. Last accessed: January 2011, 2008.

[5] J. L. Hellerstein, D. Yixin, P. Sujay, and M. T. Dawn, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[6] D. A. Menasce and M. N. Bennani, "On the use of performance models to design self-managing computer systems," in *Proc. 2003 Computer Measurement Group Conf*, 2003, pp. 7–12.

[7] A. Kamra, V. Misra, and E. M. Nahum, "Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites," in *IWQoS*, 2004, pp. 47–56.

[8] P. L. Fontaine Rafamantanantsoa and A. Aussem, "Analyse, modélisation et contrôle en temps réel des performances d'un serveur web," LIMOS, Tech. Rep. LIMOS/RR-05-06, 10 Mars 2005.

[9] M. Litoiu, "A performance analysis method for autonomic computing systems," *TAAS*, vol. 2, no. 1, 2007.

[10] T. Begin, A. Brandwajn, B. Baynat, B. E. Wolfinger, and S. Fdida, "Towards an automatic modelling tool for observed system behavior," in *In proceeding of the 4th European Performance Engineering Workshop (EPEW 2007)*, Springer, Ed. Berlin, Germany: Lecture Notes in Computer Science, 27–28 September 2007, pp. 200–212.

[11] D. A. Menascé, "Computing missing service demand parameters for performance models," in *Int. CMG Conference*, 2008, pp. 241–248.

[12] C. M. Woodside, T. Zheng, and M. Litoiu, "Performance model estimation and tracking using optimal filters," *IEEE Transactions on Software Engineering*, vol. 34, no. 3, pp. 391–406, 2008.

[13] B. Dillenseger, "Clif, a framework based on fractal for flexible, distributed load testing," *Annals of Telecom*, vol. 64, no. 1-2, pp. 101–120, Feb. 2009.

[14] L. Kleinrock, *Queueing Systems*. New York: Wiley-Interscience, 1975.

[15] R. K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and modelling*. Canada: John Wiley and Sons, Inc, April 1991.

[16] W. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton: Princeton University Press, 1994.

[17] Chakravarti, Laha, and Roy, "Kolmogorov-smirnov test," *Handbook of Methods of Applied Statistics*, pp. 392–394, 1967.

[18] R. D. Smith, "The dynamics of internet traffic: Self-similarity, self-organization, and complex phenomena," 2008, last accessed: January 2011. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:0807.3374

[19] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing networks and Markov chains. modelling and Performance Evaluation with Computer Science Applications*. Canada: JOHN WILEY and SONS, 2006.

[20] R. A. Marie, "An approximate analytical method for general queueing networks," *IEEE Trans. Softw. Eng.*, vol. 5, no. 5, pp. 530–538, 1979.

[21] I. F. Akyildiz and A. Sieber, "Approximate analysis of load dependent general queueing networks," *IEEE Trans. Softw. Eng.*, vol. 14, no. 11, pp. 1537–1545, 1988.

[22] The MathWorks, Inc., "MATLAB and simulink for technical computing," http://www.mathworks.fr/. Last accessed: January 2011, 1994-2010.

[23] D. of Statistics and Mathematics, "The R project for statistical computing," 2003, http://www.r-project.org.

[24] OW2 Consortium, "JONAS, java open application server," http://wiki.jonas.ow2.org/xwiki/bin/view/Main/WebHome. Last accessed: January 2011.

[25] ANR SelfXL project, "Selfxl - self-management of complex and large scale systems," http://selfxl.gforge.inria.fr/dokuwiki/doku.php. Last accessed: January 2011.