Assessing SLA Compliance from Palladio Component Models

Juan F. Pérez Giuliano Casale
Department of Computing
Imperial College London,
London, UK
Email: {j.perez-bernal, g.casale}@imperial.ac.uk

***Abstract*—Service providers face the challenge of meeting service-level agreements (SLAs) under uncertainty on the application actual performance. The performance heavily depends on the characteristics of the hardware on which the application is deployed, on the application architecture, as well as on the user workload. Although many models have been proposed for the performance prediction of software applications, most of them focus on average measures, e.g., mean response times. However, SLAs are often set in terms of percentiles, such that a given portion of requests receive a predefined service level, e.g., 95% of the requests should face a response time of at most 10 ms. To enable the effective prediction of this type of measures, in this paper we use fluid models for the computation of the probability distribution of performance measures relevant for SLAs. Our models are automatically built from a Palladio Component Model (PCM) instance, thus allowing the SLA assessment directly from the PCM specification. This provides an scalable alternative for SLA assessment within the PCM framework, as currently this is supported by means of simulation only.**

***Keywords*-Service-level agreements, Application Performance, Palladio Component Models**

I. Introduction

Quality of Service (QoS) plays a central role in the effective delivery of IT services, and its degradation can become a serious drawback for the service provider. In many cases the QoS offered must comply with strict service-level agreements (SLA), which impose limits on specific performance metrics. Cloud computing offers great flexibility for application deployment, as the required resources can be used on a pay-as-you-go basis. This flexibility allows the effective management of QoS for cloud applications, but its correct administration poses a number of issues. The service provider faces the problem of determining how many instances or virtual machines to buy from the cloud provider, a decision that has a direct impact on the user QoS experience, on the SLA fulfilment, and on the provider's operational costs. Further, the resource requirements highly depend on the usage profile and the hardware on which the application is deployed, factors not necessarily under the control of the service provider. To cope with this complex scenario, service providers can rely on predictive models to determine the resources necessary to offer a predefined QoS.

Typically, SLAs are specified as a percentile, such that certain portion of the requests must be satisfied without surpassing a certain value for a QoS measure, e.g. 95% of the requests must be completed in at most 10 ms. Although a number of performance models have been proposed to assess the performance of software applications, these usually rely on methods that provide average values only, e.g., the average response time experienced by the requests. To bridge the gap between average and percentile performance measures, in this paper we investigate a fluid multi-class queueing network (QN) model for the performance analysis of software applications. The advantage of the fluid QN model lies on its ability to approximate the cumulative distribution function (CDF) of the performance measures, which contains the percentile information necessary to evaluate whether an SLA is met or not. Furthermore, we have devised an automatic transformation to obtain the fluid QN model from a Palladio Component Model (PCM) [1] instance. PCM is a component-based software engineering tool that allows the description of the application architecture, its usage, and the characteristics of the hardware on which it is deployed. As a result, it is possible to obtain the performance prediction for the application directly from the PCM description, without any additional specification. Recent extensions [2], [3] to the PCM enable it to model and assess the cost and performance of cloud applications. In particular, [3] extends PCM to handle time-varying conditions typical of cloud deployments, while [2] extends PCM to model cloud infrastructures and availability requirements on cloud applications.

PCM models can be specified with the Palladio Bench tool, which already offers tools for performance prediction by means of simulation and analytic models. Although accurate and able to assess SLAs, simulation-based analysis is highly time-consuming, limiting the analysis when a large parameter space needs to be explored, such as when defining the right deployment size. In terms of analytic models, the PCM tool can map to the Layered Queueing Network (LQN) paradigm [4], which has been proven successful for the performance assessment of software applications. The advantage of LQNs is that they extend traditional QNs to handle a number of features relevant for software applications, such as resource pooling, synchronous calls between components, admission control, among others. This is done by building a set of related QN models, each one

making up a layer of the whole LQN model. An LQN model can be solved with the LQN solver (LQNS) [5], which automatically computes a number of relevant performance measures. LQNS however provides average measures only, which cannot be directly used to assess percentile SLAs. As mentioned above, our approach aims at computing the performance measures CDF, enabling the direct assessment of percentile SLAs. As its computation requirements are similar to those of LQNS, our model becomes an scalable alternative for SLA assessment, compared to more computationally intensive simulations. The fluid model we introduce, though, relies on a single QN model. The results in this paper show that, for applications that can be adequately described with a single QN model, the fluid analysis proposed is well-suited for the direct assessment of percentile SLAs. Future work will aim at extending the current fluid model to capture many of the features supported by the LQN framework.

We have implemented the modeling framework introduced in this paper in a tool called LINE, publicly available at www.doc.ic.ac.uk/~gcasale/line/. The tool automatically builds the performance model from a PCM instance, and computes relevant performance measures for SLA assessment. It is also important to highlight that the fluid model presented here is readily able to handle systems with a large number of users and servers. This ability arises from the definition of the fluid model itself, where the number of users and servers are represented as a continuous quantity. This avoids the scalability problems associated with the state-space explosion typical of Markov chain models.

After a review of related work in the next section, we introduce the transformation from a PCM instance to the QN model in Section III. The analysis of the fluid QN model is presented in Section IV, and its results are illustrated in Section V. Conclusions follow in Section VI.

II. Related Work

The Palladio Component Model (PCM) [1] is a component-based software engineering meta-model that has been designed to explicitly handle QoS aspects of software applications. This is achieved by allowing the parametric characterization of the hardware on which the application will be deployed, as well as a usage model specifying the workload characteristics. Thanks to the component-based design, these characteristics can be specified independently of the application architecture, but are related through an allocation model. PCM models are specified with the Palladio Bench tool, which offers simulation-based performance analysis via the built-in Simucom tool. Simulation analysis is both accurate and well suited for SLA assessment, as it estimates the histogram of relevant performance metrics. However, simulation is highly time-consuming when exploring a large parameter space, as when specifying the deployment characteristics at design time.

Palladio Bench also supports analytic models for performance analysis by means of LQN models [4], [6], which are an extension of queueing networks [7] that have been shown to provide an adequate framework to model software systems [8]. Palladio Bench automatically generates the LQN model of a PCM instance using the PCM2LQN model transformation [9], and the resulting model is solved with the LQN Solver (LQNS) [5], a tool for solving LQN models using approximate mean-value analysis [10] .

Other approaches rely on LQN models to compute average performance measures, but these are derived from different application models. UML4SOA models are used in [11], augmented with a MARTE [12] profile to specify timings associated to the deployment, while the Component-Based Modeling Language (CBML) is used in [13] to specify the application architecture in terms of components. Other methodologies for performance analysis of component-based software systems [14], [15] rely on traditional QN models, from which it is also possible to obtain average performance metrics. A recent survey on the topic can be found on [16].

The closest approach to our performance model is probably the fluid model for LQNs based on stochastic process algebras recently proposed in [17]. The results in [17] focus on average performance metrics, and on the good accuracy obtained with the fluid model. However, the translation from the LQN to the PEPA model makes no use of the scheduling policy defined for the processors or tasks. As the processor's scheduling policy has a significant impact on the fulfilment of SLAs, our model considers processor-sharing (PS) scheduling explicitly. The analysis of First-Come-First-Served (FCFS) processors will be the topic of future work.

III. Background

In this section we describe how a QN model can be derived from a PCM instance. To derive the QN model from the PCM instance, we proceed in two steps. The first step consists of using the PCM2LQN transformation [9] to translate the PCM application description into an LQN model. The second step derives the QN model from the LQN description. We therefore focus on the second step, starting with a general definition of QNs and LQNs, and then proceeding on how the elements of the latter can be used to parameterize the former.

A. Class-switching queueing networks

Our modeling tool is a closed multi-class queueing network (QN) with class switching [7], also known as a multi-chain QN [7]. This type of network is composed of a set of M stations and K job classes. There are a total of N jobs (or users) circulating in the network, demanding service from the servers in the stations depending on their class. Each of the N jobs belongs initially to a given class. A class-k job in station i demands an exponentially-distributed service time with rate $\mu_{i,k}$, a service that is provided by one

Table I
QN NOTATION

Parameter	Definition
M	Number of stations
K	Number of classes
N	Number of users
P	Routing probability matrix
$P_{(i,k),(j,l)}$	Probability that a class-k job finishes at station i and proceeds to station j as a class-l job
$\mu_{i,k}$	Service rate of class-k jobs at station i
m_i	Number of servers at station i
$x(t)$	State of the fluid QN at time t
$x_{i,k}(t)$	Number of class-k jobs at station i at time t
$x_i(t)$	Total number of jobs in station i at time t
$\bar{x}$	Fixed point of the ODE system
$f(x,i,k,j,l)$	Transition rate of a class-k job finishing at station i and proceeding to station j as a class-l job
$Q_{i,k}$	Average number of class-k jobs in station i
$X_{i,k}$	Average throughput of class-k jobs in station i
$U_{i,k}$	Utilization posed by class-k jobs in station i
$R_{i,k}$	Mean response time of class-k jobs in station i
W_k	Overall response time of class-k jobs

of the m_i servers in the station. The value of m_i can be a positive integer, or $+\infty$, denoting a pure delay station. After completing its service in station i, the job switches to class l and proceeds to station j with probability $P_{(i,k),(j,l)}$. Table I summarizes the notation for our reference QN model.

B. Layered queueing networks

An LQN is composed of *tasks*, which represent the servers, and are deployed on *processors*. Both tasks and processors have a multiplicity attribute, that allow the specification of multi-threaded and multi-core architectures. They also have a scheduling policy to define how the incoming requests are served. A task exposes a set of services, called *entries*, which can be called from other tasks. A simple example is shown in Figure 1, where the Web Server task, deployed on the Web Server Processor, exposes two entries: Logout and Login. Each entry has a set of *activities*, which are executed according to an activity execution graph. An activity executes by posing a demand on the processor it is deployed on, or by generating a call to an entry in a different task. These two alternatives correspond to internal and external actions in the PCM model, respectively. Apart from activities executing sequentially, a node in the activity graph can be a *probabilistic OR*, or a *fork-join* node. A probabilistic OR has a number of outgoing vertices, each with an associated label that defines the probability with which the activities in that path are executed. A fork node allows the parallel execution of the next activities, and the join node is executed only after all the previous activities have been completed. This is illustrated in Figure 1, where the Login entry executes by first calling the Query entry in the Database task, and then proceeds with a probabilistic

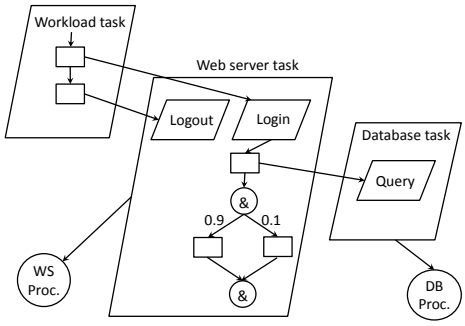


Figure 1. LQN example

OR, executing either of two activities, with probabilities 0.9 and 0.1. The call to an entry can be synchronous or asynchronous, depending on whether the caller processor is blocked until the called entry finishes or not. The workload is modelled by a *reference task* which exposes no entries, but has a multiplicity equal to the number of users. A set of activities, and their activity graph, is defined at the task level to specify how the services exposed by other tasks are called by the users. The Workload task in Figure 1 is the reference task, and sequentially calls the two entries in the Web Server task. While these are the basic components of an LQN, a number of extensions have been added to the framework, as surveyed in [4]. Currently, our model supports probabilistic OR nodes, but not fork-and-join nodes. Offering support for these and other features will be the topic of future work.

C. The transformation

We start by defining the population characteristics of the queueing network, which are taken from the reference task in the LQN. The multiplicity of this task is used as the total number of N users issuing jobs that circulate in the network. From the activity graph in the reference task we also define the number of classes K. Currently we support two types of activity graphs at this level: a single trunk, or a probabilistic OR (branch). In the former, the network is assumed to have a single class of users, while in the latter we assume there are as many classes as branches in the probabilistic OR. Further, as the reference task defines a mean think time for the users, we translate this into a delay node for the queueing network and set the mean think time as indicated in the task. The example in Figure 1 shows a single class of users, as there is a single trunk in the activity graph of the Workload task.

The next step is to follow the set of calls posed by the users in the task activities of the reference task. Each of these activities is a call to a service (entry) exposed by another task. When calling an entry, its activities will be executed according to its activity graph. At this (entry) level, we support general activity graphs, where each activity can itself call an entry in another task in a recursive fashion. This process terminates eventually, and the user then enters

its think time. For instance, in Figure 1 the user first calls the Login service, which itself calls the Query entry in the Database task. After this, one of two activities is executed, both posing a demand on the Web Server Processor. This terminates the execution of the first activity on the Workload task, so the second activity starts, which calls the Logout entry in the Web Server task. After this call is executed, the user enters a think time before submitting a new request.

This behavior is mapped into a QN model as follows. Every time an external call is made (an activity calls the execution of an entry in another processor) a new link is created between the two processors for this job class, by setting $P_{(i,k),(j,k)} = 1$, where i is the index of the processor where the caller activity is deployed, j is the index of the processor where the called entry is deployed, and k is the job class, as set in the reference task. Whenever an activity poses an actual demand on the processor i it is deployed on, the service rate $\mu_{i,k}$ is set equal to the inverse of the mean service time, where k is the job class as defined in the reference task. The multiplicity of the processor is taken as the number of servers m_i. Although in a PCM it is possible to define the processor scheduling policy as either FCFS or PS, we will assume that all the nodes, except the delay node, follow a PS policy. The extension to FCFS is left as future work. In the example in Figure 1, there are two processing stations, one for the web server, and one for the database. The user first goes to the database station, and then to the web server station, completing the execution of the first activity in the Workload activity graph. It then visits the web server station once more to complete the second activity. After this, the user visits the delay node, where it spends a think time before submitting a new request.

After executing all its activities, the user starts its think time. We model this by linking the last used processor with the delay node, setting $P_{(i,k),(\text{delay},l)} = p_l$, where i is the index of the last processor visited by the request, k is its class, and p_l is the probability that this user becomes of class l, as defined in the probabilistic OR in the reference task. This is the first appearance of the class-switching feature of the model, and allows us to mix all the requests in the delay node, and modify their class when submitting a new request. This describes well the PCM usage model, where all the users share a think time, after which they generate a new class-l request with probability p_l.

IV. Fluid Analysis

In this section we describe how to analyze the QN model, obtained by transforming the LQN description of the PCM instance, in order to compute its transient and steady-state performance measures. A simple approach would be to treat it as a product-form queueing network, for which there are well-established methods, such as the linearizer algorithm [10]. This is in fact how LQNS analyses each of the sub-models corresponding to the layers in the LQN model. We however take a different approach, by defining a fluid multi-class QN with class switching. A class-switching QN, also known as multi-chain QN [7], allows a user to change its class after completing its service at a station. This feature is key in modeling the PCM instance, as described at the end of the last section and in Section IV-D below. The fluid model allows us to compute both transient and steady-state performance measures, and it also let us compute the response time distribution, and not just its expected value, which can be effectively used to assess the fulfilment of percentile SLAs. Finally, the fluid model allows extensions that cannot be directly analyzed by means of standard methods for product-form queueing networks, and which will be the focus of future work. The notation to be introduced is summarized in Table I.

A. The class-switching PS fluid model

In our context, a fluid model is a continuous-time dynamical system described by a set of ordinary differential equations (ODE) that approximates the evolution of a stochastic system with a Markovian description. In our case, the stochastic system is a closed class-switching queueing network, the state of which at time t is given by the vector $x(t) = \{x_{i,k}(t), 1 \leq i \leq M,\ 1 \leq k \leq K\}$, where $x_{i,k}(t)$ is the number of class-k jobs in station i at time t. From now on the double index (i,k) for any vector refers to its entry $(i-1)K+k$, that is, the one corresponding to station i and class-k jobs. The system state is modified by events, which in our closed queueing network are limited to service completions. Once the service of a class-k job terminates in station i, the job proceeds to station j as a class-l job with probability $P_{(i,k),(j,l)}$. This means that the state changes from state x to state $x + \boldsymbol{e}_{j,l} - \boldsymbol{e}_{i,k}$, where $\boldsymbol{e}_{i,k}$ is a vector of zeros with a one in entry (i,k). This transition occurs with a rate proportional to $\mu_{i,k}$, as it is a class-k service completion in station i. Further, as we assume that the stations follow a PS policy, the service rate is equally divided between all the jobs present at the station, that is among $x_i = \sum_{l=1}^{K} x_{i,l}$. If we assume a single server, the transition rate $f(x,i,k,j,l)$ from state x to state $x + \boldsymbol{e}_{j,l} - \boldsymbol{e}_{i,k}$ is given by

$$f(x,i,k,j,l) = \mu_{i,k} P_{(i,k),(j,l)} \frac{x_{i,k}}{x_i} \mathbf{1}\{x_i > 0\},$$

where $\mathbf{1}\{\}$ is the indicator function. This expression poses some numerical issues due to the discontinuity at $x_{i,k} = 0$ when $\sum_{l=1, l\neq k}^{K} x_{i,l} = 0$, so we modify it as

$$f(x,i,k,j,l) = \begin{cases} \mu_{i,k} P_{(i,k),(j,l)} \frac{x_{i,k}}{x_i}, & x_i > 1, \\ \mu_{i,k} P_{(i,k),(j,l)} x_{i,k}, & 0 < x_i \leq 1, \\ 0, & x_i \leq 0. \end{cases}$$

This expression eliminates the (first-order) discontinuities, as the first and second expressions coincide when $x_i = 1$, while the second and third coincide when $x_i = 0$.

Although product-form QNs accept single PS processors only, the previous expression for the transition rates can be generalized for the multi-server case. We assume that a new job is always assigned to an idle server if any, while the m_i servers are fully shared when all of them are busy, such that the transition rates become

$$f(x,i,k,j,l) = \begin{cases} \mu_{i,k} P_{(i,k),(j,l)} \frac{x_{i,k}}{x_i} m_i, & x_i > m_i, \\ \mu_{i,k} P_{(i,k),(j,l)} x_{i,k}, & 0 < x_i \leq m_i, \\ 0, & x_i \leq 0. \end{cases}$$

For the delay stations, the number of servers is set equal to $N = \sum_{i=1}^{M} \sum_{k=1}^{K} x_{i,k}$, such that the first expression is never activated, and the transition rates are simply equal to $\mu_{i,k} P_{(i,k),(j,l)} x_{i,k}$. Therefore, this rate function considers both PS and delay stations. A more compact representation of the previous expression is

$$f(x,i,k,j,l) = \mu_{i,k} P_{(i,k),(j,l)} \frac{x_{i,k}}{x_i} \min\{m_i, x_i\}. \quad (1)$$

Since a transition with rate $f(x,i,k,j,l)$ causes the state to jump from x to $x - \boldsymbol{e}_{i,k} + \boldsymbol{e}_{j,k}$, the set of ODEs that approximate the queueing network evaluated at a point x is

$$\frac{dx}{dt} = \sum_{i=1}^{M} \sum_{j=1}^{M} \sum_{k=1}^{K} \sum_{l=1}^{K} f(x,i,k,j,l)(\boldsymbol{e}_{j,l} - \boldsymbol{e}_{i,k}).$$

Since the rate functions f are Lipschitz continuous [18], the system of ODEs has a unique solution $x(t)$.

We use $x(t)$ to approximate the actual behavior of the QN described by $X^N(t) = \{X^N_{i,k}(t), 1 \leq i \leq M,\ 1 \leq k \leq K\}$, where $X^N_{i,k}(t)$ is the number of class-k jobs in station i at time t in a QN with a population of N jobs. Starting with [19], it has been recognized that, under certain conditions, the evolution of a stochastic system with a finite population N can be approximated by a system of ODEs. Further, this approximation becomes exact when the population N approaches ∞. For closed QNs with PS stations, it has been recently shown [20] that this result holds.

B. Steady-state performance measures

Let $\bar{x}$ be a fixed point of the ODE system, found by letting the system evolve until the derivative becomes negligible. Let $Q_{i,k}$, $X_{i,k}$, and $R_{i,k}$ be the mean number, throughput, and response time of class-k jobs in station i, for $1 \leq i \leq M$ and $1 \leq k \leq K$. Also, let X_k be the throughput of class-k jobs with respect to a specific node. This is a delay node, labeled D, and represents the users think time between the completion of a request and the issue of the next one. We assume all classes visit node D, where they experience an exponential delay with mean $R_{D,k}$, for $1 \leq k \leq K$. We assume the matrix $P = [[P_{(i,k),(j,l)}]^K_{k,l=1}]^M_{i,j=1}$ can be decomposed into n_C chains or strongly connected components. These chains divide the K classes into n_C disjoint sets $\{C_q, 1 \leq q \leq n_C\}$, such that jobs can switch among classes in the same chain only.

From the definition of the state vector $x(t)$, and its fixed point $\bar{x}$, the mean number of jobs per class and station is readily available as $Q_{i,k} = \bar{x}_{i,k}$, for $1 \leq i \leq M$ and $1 \leq k \leq K$. Then, for each class we determine the throughput at node D by Little's law

$$X_{D,k} = \frac{Q_{D,k}}{R_{D,k}},\ 1 \leq k \leq K,$$

since the mean response time $R_{D,k}$ is equal to the mean delay time at node D. Now, for each chain q we determine its throughput at station D by summing up the throughput of all the classes that belong to this chain, that is

$$\widetilde{X}_{D,q} = \sum_{k \in C_q} X_{D,k},\ 1 \leq q \leq n_C.$$

To determine the throughput for each chain in the remaining stations we must first compute the average number of visits to each station per visit to station D. To this end we compute the invariant measure γ of matrix P, for which a multi-chain analysis is necessary. That is, first the n_C chains are identified, and then for each chain an invariant measure is obtained. The average number of visits to station i per visit to station D for jobs in chain q is then given by

$$\widetilde{L}^q_{i,D} = \frac{\sum_{k \in C_q} \gamma_{i,k}}{\sum_{k \in C_q} \gamma_{D,k}},\ 1 \leq q \leq n_C,$$

that is, we count visits from all the job classes belonging to chain q. The throughput of chain q at station i is therefore

$$\widetilde{X}_{i,q} = \widetilde{L}_{i,D} X_{D,k},\ 1 \leq q \leq n_C.$$

To compute the throughput for each job class, we first determine the proportion of visits to station i by jobs in chain q that corresponds to class-k jobs, referred to as $\alpha_{i,k}$. This proportion is obtained from the invariant measure γ as

$$\alpha_{i,k} = \frac{\gamma_{i,k}}{\sum_{k \in C_q} \gamma_{i,k}},\ 1 \leq i \leq M,\ 1 \leq q \leq n_C.$$

The throughput of class-k jobs at station i is then given by $X_{i,k} = \widetilde{X}_{i,q} \alpha_{i,k}$, for $1 \leq i \leq M$ and $k \in C_q$, $1 \leq q \leq n_C$. The response times are obtained by Little's law as

$$R_{i,k} = \frac{Q_{i,k}}{X_{i,k}},\ 1 \leq k \leq K,\ 1 \leq i \leq M,$$

and the utilization at station i is given by

$$U_i = \sum_{k=1}^{K} \frac{X_{i,k}}{\mu_{i,k}},\ 1 \leq i \leq M. \quad (2)$$

1) Response time distribution: Up to this point, we have focused on mean performance measures, which can also be computed with traditional methods. The fluid model is also able to provide the probability distribution of these measures, which can be used to assess the fulfilment of SLAs. Here we illustrate how to compute the total response time for class-k jobs, including the visits to all stations between two

successive visits to the delay node. In traditional QNs, the class-k average response time is obtained by considering the QN with one class-k job less, but this analysis cannot be performed here due to the fractional nature of the fluid model. Our approach however is similar in that we remove a small amount ϵ from the class-k fluid and assign it to a new class, which behaves almost identically to class-k.

Specifically, we consider the QN in steady-state, that is, we take the fixed point $\bar{x}$ as the initial state for the QN. Next, we modify the QN by adding a new job class $K+1$. The mean service times and routing matrix of this class are set equal to those of class k, with the difference that after reaching the delay node, no fluid is allowed to exit, that is $P_{(D,K+1),(D,K+1)} = 1$. Now, let S_k be the set of stations visited by class-k jobs immediately after leaving the delay node, and let q_i be the associated probability of this transition. From the fixed point $\bar{x}$, we take a portion ϵ of the class-k fluid in the delay node, and assign it to the class $K+1$ fluid, splitting it among the stations in S_k according to the probabilities p_i. In other words, we first modify the vector $\bar{x}$ to account for the extra class, adding zeros to the entries corresponding to the new class. We then build a new vector $\widetilde{x}$ as $\widetilde{x} = \bar{x} + \delta$, where the non-zero entries of δ are

$$\delta_{D,k} = -\epsilon \bar{x}_{D,k}, \text{ and } \delta_{i,K+1} = \epsilon \bar{x}_{D,k} q_i,$$

for $i \in S_k$. We now solve the modified system of ODEs in Eq. (1) with the additional class, and with initial vector $\widetilde{x}$, and let it evolve until $x_{D,K+1}$, the class-$K+1$ fluid in the delay node, reaches ϵ. The response time distribution is approximated by the path $x_{D,K+1}(t)/\epsilon$, i.e., if W_k denotes the random variable of the response time for class-k jobs,

$$P(W_k \leq t) \approx \frac{x_{D,K+1}(t)}{\epsilon}, \ t \geq 0.$$

C. Transient performance measures

Computing the transient performance measures is similar to obtaining the steady-state ones. Solving the ODE, the whole history of $x(t)$ is available, from $t = 0$ to any time $t = T$, such that the expected queue-length at a time t is

$$Q_{i,k}(t) = x_{i,k}(t), \ 1 \leq i \leq M, \ 1 \leq k \leq K.$$

To compute the expected throughput of class-k jobs in station i, $X_{i,k}(t)$, we obtain the effective rate at which jobs are being processed given the current queue-lengths, as

$$X_{i,k}(t) = \sum_{j=1}^{M} \sum_{l=1}^{K} \frac{f(x(t), i, j, k, l)}{P_{(i,k),(j,l)}},$$

for $1 \leq i \leq M, \ 1 \leq k \leq K$. From this we can obtain the expected utilization at time t by means of Eq. (2), replacing $X_{i,k}$ by its transient version $X_{i,k}(t)$.

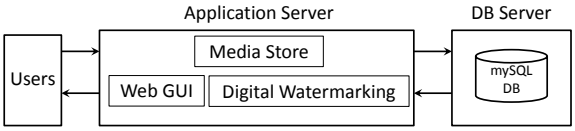


Figure 2. Media store application

D. Artificial classes

As described in Section III, when building the QN model from the LQN description of a PCM instance, the class of a job is defined by the branch, if any, in the reference task. However, to fulfil its service, the job may need to visit a certain processor i several times, *each with a different service time*. An initial option to handle this behavior is to average the service times at each visit. As the node visited after node i may be different in each visit, the exit probabilities need to be averaged as well. To avoid losing routing and service time information by averaging, we consider a different alternative where a second visit to node i is represented by making the class-k job switch to a new class k', which inherits the subsequent path and service times from class k. When the class-k' job visits a node for a second time, the process is repeated, creating a new class and forcing the class-k' job to switch to this new class. In this manner, for each original class k, a set of artificial classes $K(k)$ is created.

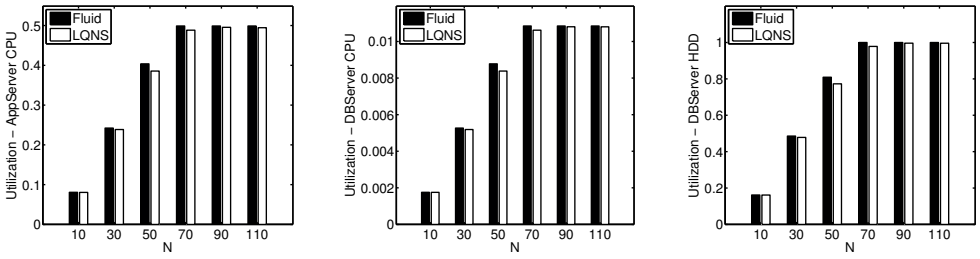
Using the disjunct class sets $K(k)$, we obtain a network description with a larger set of classes $\hat{K} = \cup_{k=1}^{K} K(k)$. To compute the performance metrics for the original classes, we simply add the mean numbers and throughputs for the corresponding artificial classes, as

$$Q_{i,k} = \sum_{l \in K(k)} \hat{Q}_{i,l}, \ X_{i,k} = \sum_{l \in K(k)} \hat{X}_{i,l},$$

for $1 \leq i \leq M, \ 1 \leq k \leq K$, where $\hat{Q}_{i,l}$ and $\hat{X}_{i,l}$ are the metrics obtained for the queueing network with the artificial classes. From these we can compute the mean response times and utilizations for the original network as in the previous section. Notice that a larger number of classes means a larger number of ODEs, which implies longer computation times to obtain the performance measures.

V. Results

In this section we illustrate the behavior of the fluid performance model, and compare its results against LQNS. We make use of the Media Store application [1], the architecture of which is depicted in Figure 2, and whose components include a web GUI, a store manager component, a watermaking component, and a mySQL database. The database is deployed on a database server, while the other components are deployed on the application server. These components, their interaction, and the deployment characteristics are specified in a PCM [1]. The key resources



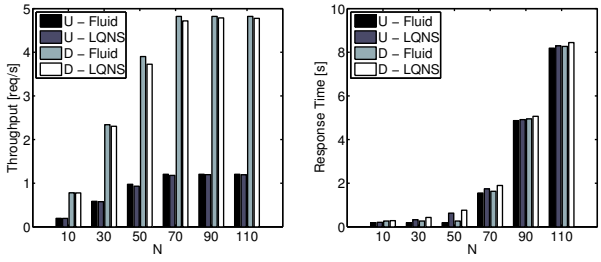
(a) App. server CPU (b) DB server CPU (c) DB server HDD

Figure 3. Server utilization for the Media Store application

for the deployment are the CPU of the servers, and the disk (HDD) of the DB server. There is a closed population of N users, whose requests can be to download or upload files. For the experiments presented here, we have eliminated a connection pool associated to the database, as this imposes a finite capacity region in the QN that the current version of the fluid model cannot handle. For the same reason, we also set the scheduling policy of all the processors to PS. The model was implemented and solved in MATLAB 2012b, making use of its ODE solvers. The experiments reported here were obtained on a 3.4 GHz 4-core Intel Core i7 machine, with 4GB RAM, running Linux Ubuntu release 12.04. With this configuration, solving the fluid model and obtaining all the performance measures takes less than a second, times similar to those of LQNS.

1) Average performance: Figure 3 presents the predicted server utilization for population sizes between 10 and 110 users. Clearly, the application server CPU and the DB server HDD have a larger utilization than the DB server CPU, and all these are rather small for the case with 10 users. Increasing the number of users increases the utilization, but with 70 users the DB server HDD reaches its maximum capacity, and becomes the system bottleneck. As shown in Figure 4(a), increasing the user population up to 70 users increases the throughput, but beyond this point the throughput does not increase, as the system is limited by the DB server HDD. The QoS degradation when the user population reaches 70 users or more is captured in Figure 4(b), where the mean response time for both download and upload services shows a sharp increase. In a cloud deployment, this implies that more resources are needed, and a rising trend reaching the 70 users limit should trigger the start-up of a new instance of the DB server to prevent the QoS degradation. Such a trend could be spotted in the monitoring data, triggering the corresponding scale-up action.

2) Comparison with LQNS: In Figures 3-4 we present the results for both the fluid model and LQNS, showing a very good agreement between both. The only significant difference occurs in the response times for 30 and 50 users. This difference is caused by the approximation in the processing rate introduced in the fluid model. Since



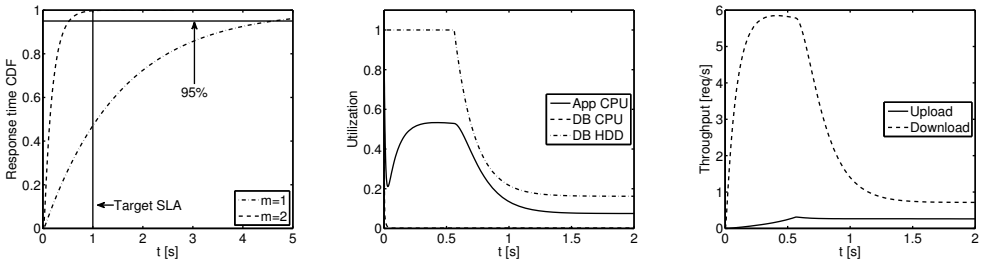
(a) Throughput (b) Response time

Figure 4. Performance of Download (D) and Upload (U) services

the approximation is introduced when the number of jobs in process is less than one, as the load increases and this threshold is crossed the response time predictions of the fluid model and LQNS become very similar.

3) Meeting SLAs: Figure 5(a) depicts the response time CDF for the download service provided by the Media Store application. This case corresponds to a population of 70 users, and we show two cases. In the first case, there is a single server for the DB processor ($m = 1$). Comparing with Figure 4(b), we observe that the expected response time for this configuration is around 2 seconds, but from the CDF we learn that for 95% of the requests we can only guarantee that their response time will be under 5 seconds. This configuration violates a hypothetical SLA where 95% of the download requests must complete in less than one second. We therefore opt for adding a new processor ($m = 2$), and under this configuration we expect 95% of the requests to be completed in less than 600 ms, complying with the SLA. This simple example illustrates how a decision about SLAs can be answered with the performance metric CDF, as computed with the proposed fluid model.

4) Transient results: Figures 5(b) and 5(c) illustrate how the utilization and the throughput behave as a function of time. Clearly, both measures variate significantly before settling on their steady-state value, an effect that pure steady-state analysis cannot capture. In a cloud deployment, these transient effects gain significant importance, as the number of resources may vary with the workload, and the actual



(a) Download Response time CDF (b) Transient utilization (c) Transient throughput

Figure 5. Response time distribution and transient measures

resource capacity can be affected by multi-tenancy.

VI. Conclusion and Future Work

The results presented in the previous section confirm the potential of the fluid model to evaluate both the transient and the steady-state performance of cloud component-based applications. Moreover, the model provides the CDF of relevant performance metrics, such as the response time, from which it is possible to assess the fulfilment of SLAs.

We have implemented the proposed modeling framework in a tool called LINE, publicly available at www.doc.ic.ac.uk/~gcasale/line/, which automatically builds and solves the fluid performance model from a PCM instance. We also plan to extend the fluid model to consider, for instance, more general service-time distributions.

Acknowledgement

The research of Giuliano Casale and Juan F. Pérez leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 318484.

References

[1] S. Becker, H. Koziolek, and R. Reussner, "Model-based performance prediction with the palladio component model," in *Proc. of the 6th WOSP*, 2007.

[2] M. Miglierina, G. P. Gibilisco, D. Ardagna, and E. D. Nitto, "Model based control for multi-cloud applications," in *Proc. MiSE*, 2013.

[3] D. Franceschelli, D. Ardagna, M. Ciavotta, and E. Di Nitto, "SPACE4CLOUD: a tool for system performance and cost evaluation of cloud systems," in *Proc. of Multi-Cloud*, 2013.

[4] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi, "Enhanced modeling and solution of layered queueing networks," *IEEE Trans. Softw. Eng.*, vol. 35, pp. 148–161, 2009.

[5] G. Franks, P. Maly, M. Woodside, D. C. Petriu, A. Hubbard, and M. Mroz, *Layered Queueing Network Solver and Simulator User Manual*, Carleton University, 2013.

[6] J. Rolia and K. Sevcik, "The method of layers," *IEEE Trans. Softw. Eng.*, vol. 21, pp. 689–700, 1995.

[7] F. Baskett, K. Chandy, R. Muntz, and F. Palacios, "Open, closed and mixed networks of queues with different classes of customers," *J. Assoc. Comput. Mach.*, vol. 22, pp. 248–260, 1975.

[8] G. Franks and M. Woodside, "Multiclass multiservers with deferred operations in layered queueing networks, with software system applications," in *Proc. of the 12th IEEE MASCOTS*, 2004.

[9] H. Koziolek and R. Reussner, "A model transformation from the palladio component model to layered queueing networks," in *Performance Evaluation: Metrics, Models and Benchmarks*. Springer, 2008.

[10] K. M. Chandy and D. Neuse, "Linearizer: a heuristic algorithm for queueing network models of computing systems," *Commun. ACM*, vol. 25, pp. 126–134, 1982.

[11] M. Tribastone, P. Mayer, and M. Wirsing, "Performance prediction of service-oriented systems with layered queueing networks," in *Proc. of ISoLA 2010*, 2010.

[12] Object Management Group (OMG), "A UML profile for MARTE: Modeling and analysis of real-time embedded systems, beta 2," OMG, Tech. Rep. OMG Document Number: ptc/2008-06-09, 2008.

[13] X. Wu and M. Woodside, "Performance modeling from software components," in *Proc. 4th WOSP*, 2004.

[14] V. Grassi, R. Mirandola, and A. Sabetta, "From design to analysis models: a kernel language for performance and reliability analysis of component-based systems," in *Proc. of the 5th WOSP*, 2005.

[15] Y. Liu, A. Fekete, and I. Gorton, "Design-level performance prediction of component-based applications," *IEEE Trans. Softw. Eng.*, vol. 31, pp. 928–941, 2005.

[16] H. Koziolek, "Performance evaluation of component-based software systems: a survey," *Perf. Eval.*, vol. 67, pp. 634–658, 2010.

[17] M. Tribastone, "A fluid model for layered queueing networks," *IEEE Trans. Softw. Eng.*, vol. 39, pp. 744–756, 2013.

[18] J. F. Pérez and G. Casale, "A fluid model for closed queueing networks with PS stations," Imperial College London, Tech. Rep. DTR13-8, 2013.

[19] T. G. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *J. Appl. Probab.*, vol. 7, pp. 49–58, 1970.

[20] J. Anselmi, B. D'Auria, and N. Walton, "Closed queueing networks under congestion: non-bottleneck independence and bottleneck convergence," *Math.Oper. Res.*, vol. 38, pp. 469–491, 2013.