Predicting Response Times of Applications in Virtualized Environments

Henda Ben Cheikh
LAAS-CNRS
Toulouse, France
hbenchei@laas.fr

Josu Doncel
LAAS-CNRS
Toulouse, France
jdoncel@laas.fr

Olivier Brun
LAAS-CNRS
Toulouse, France
brun@laas.fr

Balakrishna Prabhu
LAAS-CNRS
Toulouse, France
bala@laas.fr

***Abstract*—We propose simple queueing models for predicting response times of applications executed in a cloud computing platform under the SaaS model. We assume that each application instance is executed within a virtual machine running on a computing node of a data-center, and that VMs running concurrently on the same node share fairly its capacity. Our main contribution is to explicitly take into account the different behaviors of the different classes of applications (interactive, CPU-intensive or permanent applications). We show that simple expressions of the mean processing times of applications can be obtained using standard results from queueing theory. Experiments on a real virtualized platform show that the mathematical models allow to predict response times accurately.**

Keywords*—*Cloud computing, Performance estimation, BCMP Network.

I. Introduction

Cloud computing is the delivery of computing resources as a service over the Internet in an on-demand fashion [1], [2]. This new approach to computing eliminates the cost of purchasing and maintaining the computing infrastructures, and enables end-users to focus on what the service provides them rather than on how the services are implemented or hosted. Indeed, cloud users just have to submit a request for computing resources, which are then supplied on-demand by the cloud provider from its large pools installed in data centers for the required period of time.

Cloud computing providers offer their services according to several fundamental models. In the model known as *Software as a Service* (SaaS) [3], cloud providers (e.g. Google Apps or Microsoft Office 365) install and operate application software in the cloud and cloud users access the software from cloud clients. Cloud users do not manage the cloud infrastructure and platform where the application runs. This eliminates the need to install and run the application on the cloud user's own computers, which simplifies maintenance and support.

Each application is usually ran in a separate virtual machine (VM) that is executed using resources (computer, storage and network devices, ...) in the data-center. A hypervisor (e.g., Xen [4] or KVM [5]) is executed on each node of the data-center and runs the virtual machines as guests. Pools of hypervisors within the cloud operational support-system can support large numbers of virtual machines and the ability to scale services up and down according to customers' varying requirements.

As should be apparent from the above, virtualization plays a key role in cloud computing solutions. A VM is just a software implementation of a machine (i.e. a computer) that executes programs like a physical machine. Although a VM is less efficient than a real machine in that it accesses the hardware indirectly, a key advantage of VMs is to allow multiple OS environments to co-exist on the same computer, in strong isolation from each other. Thus, multiple VMs each running their own operating system (called guest operating system) are frequently used in server consolidation, where different services that used to run on individual machines to avoid interference are instead run in separate VMs on the same physical machine.

However, when multiple VMs concurrently run on the same physical host, they share the available physical resources, causing unpredictable drop in their performances when some of them have compute-intensive peaks. Service providers have therefore to carefully provision the amount of resources required in order to meet user performance requirements. To this end, they need performance models in order to evaluate the performances that can be achieved from a given hardware configuration.

This paper is devoted to the performance modelling of applications executed in a cloud computing platform under the SaaS model. We assume that each application instance is encapsulated in a virtual machine which is then executed on a computing node of the data-center. The virtual machines running concurrently on a physical machine share its resources – CPU, network bandwidth, memory, etc. – in a fair way. The main modelling difficulty comes from the variety of applications that can be executed. These applications have a wide-range of resource requirements. Some such as a simulation script require resources continously over an interval of time while for others such as document edition the requirements could be scattered over an interval of time. In order to know the amount of physical resources that are in use at any given moment in time, it is thus necessary to know which applications are active and which of them are using resources. Our main contribution is to show that simple expressions of the mean processing times of applications and other performance measures can be obtained using standard results from queueing theory. Although simple, these expressions are highly accurate as shown by experimental results.

The rest of the paper is organized as follows. In Section II we put our work in the context of the existing cloud computing literature. We describe the model and derive closed-form expressions for the main performance metrics in Section III. In Section V we describe the experimental setting. We analyze the accuracy of the theoretical performance measures with respect to experimental results in Section VI. Finally, in Section VII we summarize the main contributions of the paper and discuss future works.

II. Related Work

There are relatively few works on the performance modeling of virtualized environments. In [6], the author proposes a queueing model to evaluate the performance and power consumption of systems with VMs. Although simple, the model requires a large number of parameters and may be difficult to use in practice. The authors in [7] focus on the case of virtualized system for server consolidation. They show that classical queuing theory can be helpful to predict performance metrics of virtualized systems for the studied server consolidation scenario. A queuing-theoretic approach was also used in [8] to build performance models for virtualized environments under Xen Vms.

The authors in [9] analyse the trade-off between performance and power consumption in virtualized environments. They assume a fixed number of VMs running web servers, Poisson arrivals of http requests for each VM, and a GPS scheduling discipline for the hypervisor. Using mean field theory [10], they obtain simple approximations for the mean processing time of jobs, among others. The main difference with our work is that they focus on web servers, whereas we rather consider the executions of CPU-intensive and interactive applications. The trade-off between energy and response time was also considered in [11] and [12]. In [13], an experimental evaluation on how the VM start-up time depend on various factors (time of the day, OS image size, etc.) was done for several cloud providers (Amazon EC2, Windows Azure and Rackspace).

In contrast to above works, we are mainly interested in predicting the mean processing time of applications in a virtualized environment. Our main contribution is to explicitly take into account the different behaviors of the different classes of applications (interactive or CPU-intensive applications and web servers), and to propose simple queueing formula to evaluate their performances. Experiments on a real virtualized platform shows that the mathematical models allow to predict response times accurately.

III. Model Description

As depicted in Figure 1, we consider a set of users that submit jobs for execution on the servers of a data center. We assume in the following that the population of users can be considered as infinite, with the meaning that it is large enough for the job arrival rate to be considered as independant of the number of active users[1].

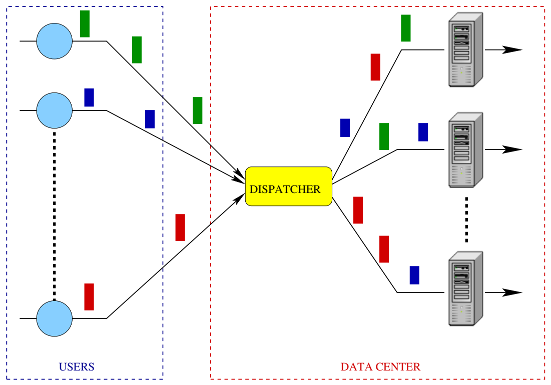

Fig. 1: Users sharing the resources of a data center.

Each individual user can be either idle (inter-session phase) or involved in an on-going session of a remote application (see Figure 2). We shall distinguish three broad categories of applications:

1) *Interactive*: Applications in this class require resources for a finite duration which could be interspersed in time with periods of inactivity. Document edition is an example of applications in this class. A user who wishes to edit a document, instanciates a document editor only when required. While editing, the user may take some time to think in between writing text. During the think-time of the user, the document editor does not require much computation resources and could be considered inactive.
2) *CPU-intensive*: This class of applications are those that require physical resources for a continuous and finite duration of time. Simulation or numerical-experiment scripts are examples of this class. Once instanciated they remain active throughout their lifetime which is of finite duration.
3) *Permanent*: Applications in this class are applications such as web servers or database servers that have no end date. Once a virtual machine hosting a web server is instanciated, it is assumed to be functional forever either waiting for requests or processing them simultaneously. Even though the virtual machine for this class will always be instantiated, its physical resource requirements may vary with the demand, i.e, a web server will need computing and memory resources only when it is processing requests. During the rest of the time, its resource requirements will be negligible.

In the sequel, we let $\mathcal{B}$ denotes the set of CPU-intensive applications, $\mathcal{I}$ be the set of interactive applications, and $\mathcal{P}$ be the set of permanent applications. We denote by $\mathcal{R} = \mathcal{B} \cup \mathcal{I} \cup \mathcal{P}$ the set of all applications available to the users, and by R the total number of applications. It is assumed that they are numbered from 1 to R.

[1]If this is not the case (e.g., for a private cloud), a similar model can be developed using the theory of closed queueing networks.

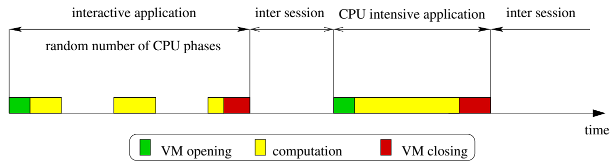

Fig. 2: Activity of an individual user accross time.

New sessions of non-permanent application are initiated by users according to a Poisson process at rate λ. This assumption stems naturally from the fact that individual sessions are independently generated by a large population of users. Upon submission, a request for application execution is received by a dispatcher, which routes the job to one of the N servers according to a (probabilistic) round-robin policy. As a consequence, new sessions are initiated on the servers according to independant Poisson processes, each one at rate λ/N, and we can analyze each server in isolation. In the following, we thus consider a single server, say server i.

When a new session of a non-permanent application is started, the server first opens a VM. We denote by $1/\mu_0(i)$ the average time required to open a VM on server i. Once the VM ready, the server starts executing the application within this VM. A new session uses application r with probability γ_r. Each execution cycle starts by a computing phase on the CPU. We let $1/\mu_r(i)$ be the mean duration of a computing phase for application r. The numbers of computing phases during a session are i.i.d. random variables, with mean $1/p_r$. By convention, $p_r = 1$ if r is a CPU intensive application, i.e., $r \in \mathcal{B}$. It is assumed that the number of computing phases by an interactive application $r \in \mathcal{I}$ is distributed according to a geometric distribution, and we let $1/\beta_r$ denote the average duration of the OFF period for this application. Finally, after the last computing phase, the VM is closed and all its ressources are freed. As before, we assume that the time it takes to close a VM on server i is a random variable, and we denote by $1/\mu_{R+1}(i)$ its mean. As well, we assume that new sessions of permanent application are initiated by users requests according to a Poisson process. It is assumed that the mean processing time of requests are random variables. In the following, it is assumed that all random variables are independant random variables.

IV. Performance metrics

The performance model introduced in Section III is very general and reveals difficult to analyze. In the following, we shall consider two special cases. In Section IV-A, we show that in the presence of both CPU-intensive and interactive applications, the system can be analyzed using standard results from queueing theory, and that simple expressions for the mean processing time of jobs can be obtained. In Section, IV-B, we consider the scenario in which a single VM running a permanent application is executed concurrently with VMs running CPU-intensive jobs. Although the analysis is far more involved than in the previous case, we obtain simple approximations for the mean processing times of jobs and requests.

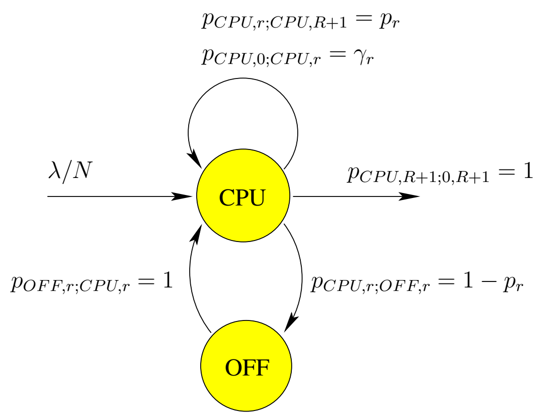
$p_{CPU,r;CPU,R+1} = p_r$
$p_{CPU,0;CPU,r} = \gamma_r$
λ/N
CPU
$p_{CPU,R+1;0,R+1} = 1$
$p_{OFF,r;CPU,r} = 1$
$p_{CPU,r;OFF,r} = 1 - p_r$
OFF

Fig. 3: Equivalent open network of queues, where $p_{s,c;s',c'}$ is the probability that a class-c customer who completes service at node s will next require service at node s' in class c'.

A. Presence of CPU-intensive and interactive applications

In this section, we study the case where there is both CPU-intensive and interactive application in the system. We observe that the execution of applications on the server can be modeled as an equivalent multiclass network of queues, as depicted in Figure 3. Customers in this queueing network represent on-going applications. Exogeneous arrivals represent new sessions started by users on server i, while departures from the network represent the end of sessions. There are $R+2$ classes of customers:

- Class-0 customers represent VMs beeing opened; all customers enter the network as class-0 customers,
- Class-r customers, where $1 \le r \le R$, represent VMs executing application r,
- Class-$R+1$ customers represent VMs beeing closed.

Each node in this queueing network is associated to a possible state of an on-going application: customers at the cpu node represent VMs beeing executed (either beeing opened, closed, or executing an application depending on the class of the customer), while customers at the off node represent VMs executing interactive applications which are in the OFF period. Since it is assumed that the VMs running concurrently on a server share its CPU in a fair way, the cpu node is modelled as a Processor Sharing node [14]. The off node is an Infinite Server queue which introduces a random delay, of mean $1/\beta_r$ for the interactive application r.

Finally, let $\rho_{cpu}(i)$ and ρ_{off} be the offered loads of the two nodes in this queueing network. We observe that the arrival rate of class-0 and class-$(R+1)$ customers at node cpu is λ/N, while the arrival rate of class-r customers at this node is $\lambda\gamma_r/(N\, p_r)$ for $r \in \mathcal{R}$. The arrival rate of class-r customers at node off is $\lambda\gamma_r(1-p_r)/(N\, p_r)$, and 0 for the two other classes. We thus easily obtain that $\rho_{cpu}(i) = \sum_{r=0}^{R+1} \rho_{cpu}^r(i)$, where

$$\begin{aligned}\rho^0_{cpu}(i) &= \frac{\lambda}{N}\,\frac{1}{\mu_0(i)}\\ \rho^r_{cpu}(i) &= \frac{\lambda}{N}\,\frac{\gamma_r}{p_r\mu_r(i)}\\ \rho^{R+1}_{cpu}(i) &= \frac{\lambda}{N}\,\frac{1}{\mu_{R+1}(i)},\end{aligned}$$

and that $\rho_{off} = \sum_{r=1}^{R}\rho^r_{off}$ where $\rho^r_{off} = \lambda\gamma_r(1-p_r)/(N\,p_r\,\beta_r)$. Our main results are stated below.

1) Joint distribution and its marginals: Let n^r_s denote the number of sessions of application r in phase $s = cpu, off$. Let us define the following vectors: $\mathbf{n}_{cpu} = \left(n^r_{cpu}\right)_{r=0,\ldots,R+1}$ and $\mathbf{n}_{off} = \left(n^r_{off}\right)_{r=1,\ldots,R}$. Let $\pi(\mathbf{n})$ be the steady-state probability that the system is in state $\mathbf{n} = (\mathbf{n}_{cpu}, \mathbf{n}_{off})$, if it exists. Finally, let the random variable N_s (resp. N^r_s) be the total number of sessions (resp. of application r) in phase s. The queueing network described above can be analyzed as a BCMP network [15], yielding the following result.

Proposition 1: The stationary joint distribution π exists provided that $\rho_{cpu}(i) < 1$, and it has the following simple product-form,

$$\pi(\mathbf{n}_{cpu}, \mathbf{n}_{off}) = \pi_{cpu}(\mathbf{n}_{cpu})\,\pi_{off}(\mathbf{n}_{off})$$

where

$$\begin{aligned}\pi_{cpu}(\mathbf{n}_{cpu}) &= (1-\rho_{cpu}(i))\,|\mathbf{n}_{cpu}|!\prod_{r=0}^{R+1}\frac{(\rho^r_{cpu}(i))^{n^r_{cpu}}}{n^r_{cpu}!},\\ \pi_{off}(\mathbf{n}_{off}) &= e^{-\rho_{off}}\prod_{r=1}^{R}\frac{(\rho^r_{off})^{n^r_{off}}}{n^r_{off}!},\end{aligned}$$

with the notation $|\mathbf{n}_{cpu}| = \sum_{r=0}^{R+1} n^r_{cpu}$. Moreover, $\Pr[N_{cpu} = k] = (1-\rho_{cpu}(i))\,(\rho_{cpu}(i))^k$ and $\Pr[N_{off} = k] = \frac{(\rho_{off})k}{k!}\,e^{-\rho_{off}}$.

Note that the above results, as well as all the results presented in this section, are insensitive to detailed application characteristics. In other words, they depend on the stochastic distributions used for computing time durations or for think times, only though their means.

2) Key performance measures: Let D (resp. D^r) be the duration of an execution (resp. of application r). We have the following results regarding the expected numbers of sessions and the mean processing times of applications.

Proposition 2: The mean number of sessions in phase *cpu* is given by

$$E[N_{cpu}] = \frac{\rho_{cpu}(i)}{1-\rho_{cpu}(i)} \qquad (1)$$

whereas $E[N_{off}] = \rho_{off}$. Moreover, the mean number of sessions of application r being executed is $E[N^r_{cpu}] = (\rho_{cpu}(i)^r/\rho_{cpu}(i))\,E[N_{cpu}]$ for $r \in \mathcal{R}$.

From Little's law, the mean response time of jobs is $D = N\,\frac{E[N_{cpu}]}{\lambda}$, while the mean processing time of application r is $D^r = N\,\frac{E[N^r_{cpu}]}{\lambda\,\gamma_r}$.

Remark 1: It was assumed for simplicity that the probability that an incoming job be routed to server i is $p_i = \frac{1}{N}$, $\forall i = 1,\ldots,N$. This assumption is natural for homogeneous servers, but it might be interesting to use a different load-balancing strategy if there are servers of different speeds. In this case, the above formulas are still valid: one just has to replace λ/N by p_i in the definitions of $\rho^r_{cpu}(i)$ and ρ^r_{off}.

B. Presence of CPU-intensive and permanent applications

In this section, we study the case where there is both CPU-Intensive and permanent applications in the system. It proves difficult to analyze the model since standard results from queueing theory cannot be used in this case. Given the complexity of the model, we shall consider the scenario in which the computer executes a single VM running a permanent application (say, a web server) concurrently with VMs running CPU-intensive applications. We shall further make a number of simplifying assumptions. First of all, we shall assume that all random variables are independant and exponentially distributed. We also assume that there is a finite number M of CPU-intensive jobs that can be executed concurrently (the maximum number of virtual processors that can be assigned to a single core is usually $M = 8$). In the following, VMs running CPU-intensive jobs shall be called class-1 jobs, while http requests submitted to the web server shall be called class-2 jobs.

Class-1 jobs are submitted by users according to a Poisson process at rate λ_1 and have a mean service time equals to $1/\mu_1$ (including the time to open and close the VM). Requests are submitted to the web server according to a Poisson process at rate $\lambda_2 \gg \lambda_1$. We denote by $1/\mu_2$ their mean service time and assume that $1/\mu_2 \ll 1/\mu_1$. The traffic intensity for class $i = 1, 2$ is $\rho_i = \lambda_i/\mu_i$. Let (n_1, n_2) be the state of the system, where n_i is the number of class-i jobs being executed. All active VMs receive a fair share of the capacity, so that in state (n_1, n_2) class-1 jobs get an aggregate service rate equals to

$$\phi_1(n_1, n_2) = \frac{n_1}{n_1 + \mathbb{1}_{\{n_2>0\}}},$$

while class-2 jobs are served with the aggregate rate $\phi_2(n_1, n_2) = 1 - \phi_1(n_1, n_2)$.

1) Quasi-stationary Assumption: We note that the steady-state distribution of the number of jobs of each class can be computed, provided it exists, by numerically solving a Markov chain. However, in order to obtain explicit performance formulas and avoid a high solving time in some cases (M is high), we shall instead analyze the system under a quasi-stationary (QS) assumption. The basic idea is that, since class-2 jobs arrive at a higher rate and have a far lower processing time, the dynamic of the number of class-2 jobs is far more faster than that of class-1 jobs, suggesting that the number of class-2 jobs can reach a statistical equilibrium before the number of class-1 jobs has evolved. The approach is thus to compute the steady-state distribution $\pi(n_2|n_1)$ of the number of class-2 jobs, given the number of class-1 jobs. Once this conditional distribution is known, the stochastic process $n_1(t)$ can be analyzed independantly by assuming that when there are n_1 jobs of class-1, they are served with the average service rate $\tilde{\phi}_1(n_1) = \sum_{n_2} \phi_1(n_1, n_2)\pi(n_2|n_1)$. Following this approach, we obtain Proposition 3.

Proposition 3: Provided that $\rho_2 < \frac{1}{1+M}$, the QS distribution of the number of class-2 jobs exists and is given by

$$\pi(n_2|n_1) = (1 - (1 + n_1)\rho_2)\ (1 + n_1)^{n_2}\ \rho_2^{n_2}, \tag{2}$$

for $n_2 = 0, 1, \ldots$ The steady-state probability that there $n_1 = 0, 1, \ldots, M$ jobs of class-1 being executed is given by

$$\pi_1(n_1) = \begin{cases} \frac{1}{1+M} & \text{if } \nu = 1, \\ \frac{1-\nu}{1-\nu^{1+M}}\ \nu^{n_1} & \text{if } \nu \neq 1, \end{cases} \tag{3}$$

where $\nu = \rho_1/(1 - \rho_2)$.

Proposition 3 immediately yields the following corollary.

Corollary 1: The steady-state average number of class-1 jobs is $\mathbb{E}(N_1) = \frac{M}{2}$ if $\nu = 1$, and

$$\mathbb{E}(N_1) = \frac{\nu}{1-\nu} - (1 + M)\frac{\nu^{1+M}}{1-\nu^{1+M}}, \tag{4}$$

otherwise. The steady-state average number of class-2 jobs is given by

$$\mathbb{E}(N_2) = \sum_{n_1=0}^{M} \pi_1(n_1)\, \frac{\rho_2(1 + n_1)}{1 - \rho_2(1 + n_1)}. \tag{5}$$

2) Validity of the QS Assumption: We compare below the approximation obtained under the QS assumption with the exact results obtained by solving the Markov Chain $(n_1(t), n_2(t))$. We denote by α the ratio $\frac{\lambda_1}{\lambda_2}$ and by β the ratio $\frac{\mu_1}{\mu_2}$. By assumption, $\alpha \ll 1$ and $\beta \ll 1$. We further assume that $M = 10$. Figures 4 and 5 show the evolution of the relative error on the expected number of ongoing jobs of class 1 and 2 obtained using (4) and (5) depending on the product $\alpha\beta$. We observe that the results obtained from (4) and (5) are very close to the numercical results obtained by solving the Markov chain for both class-1 and class-2 of jobs and for all considered scenarii. The relative error is less then 2% for class-1 jobs and 7% for class-2 jobs. We have also validated the approximation for different values of the product $\alpha\beta$. We note that the relative error decrease when the QS assumption is satisfied i.e. $\alpha\beta$ is small enough. The results confirm that expressions (4) and (5) constitute a good approximation.

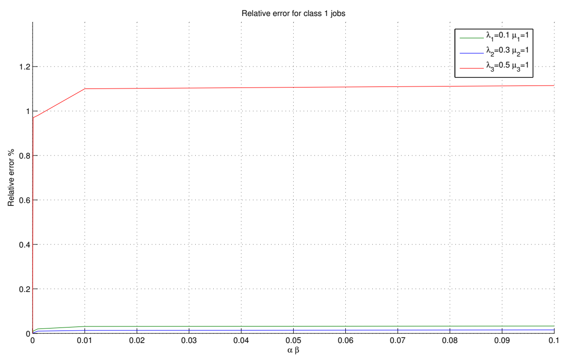

Fig. 4: Relative error for class-1 jobs.

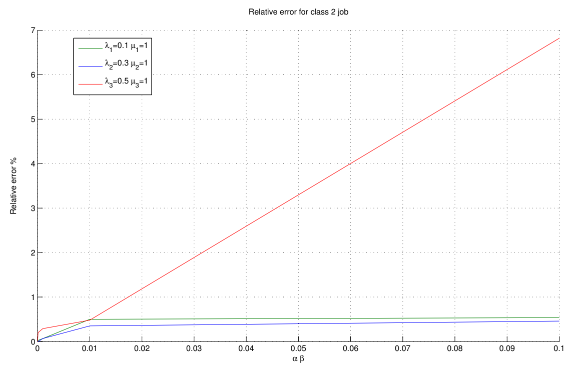

Fig. 5: Relative error for class-2 jobs.

V. Experimental setting

We have done a set of experiments to compare the predictions of the mathematical models described in Section IV against the measured behavior of the system. These experiments amount to running VMs executing CPU-intensive, interactive or permanent applications on a host computer, and measuring the mean processing time of jobs/requests. In this section, we describe the experimental setting and the emulated scenarios.

A. Host computer

The virtualized environment within which the applications were executed was installed on a quadri-processor server (Intel Xeon 2.3 GHz with 4 CPU cores per processor), 4 GB of RAM, 500 GB of hard disk capacity, and Ubuntu 10.04.4 LTS as the operating system. For virtualization, we have choosen VirtualBox [16]. We have installed 8 VMs on the host computer, running under the same Ubuntu version as the host. The memory allocated to each Virtual Machine is 512 MB (minimum value required for the Operating System).

B. Emulating Poisson Processes

New sessions of non-permanent applications are initiated on the host computer according to a Poisson process at rate λ. A Matlab program was used to generate the arrival instants $T_0, T_1, \ldots$ of the new sessions according to the following recursion:

$$T_{k+1} = T_k - (1/\lambda) * (\log(u)), k = 0, 1, \ldots \tag{6}$$

where $u \in (0, 1)$ is a uniform random number and $T_0 = 0$. Once the arrival instants generated, application executions are scheduled using cron, a time-based software for job scheduling purposes in Unix [17]. The executed scripts open a VM, execute the application within the VM, and then close the VM.

C. Scilab: CPU-intensive application

As CPU-intensive application, we have choosen a Scilab program. Scilab is a high-level, numerically oriented programming language [18]. The main applications of Scilab are signal processing, statistical analysis, numerical optimization, and modeling, simulations and (if the corresponding toolbox is installed) symbolic manipulations. Our Scilab program solves an optimization problem of mathematical ecology using dynamic programming (see [19] for details).

D. OpenOffice: interactive application

As interactive apllication, we have choosen the Spreadsheet application of OpenOffice [20]. This software will let us connect from a remote computer to a server that runs the word processor in a interactive and easy way. We wrote a JAVA program that use UNO components from the Apache OpenOffice Software Development Kit[2]. This program opens a SpreadSheet, insert some data in it then draw some 3D graphics. In order to emulate the interactivity of a user, the program alternates between ON periods, where data are entered into the spreadsheet, and OFF periods, where the application is inactive. The number of ON-OFF cycles follows a geometric distribution with mean $\frac{1}{p} = 125$. The duration of ON and OFF periods are exponentially distributed, with mean 800 ms and 200 ms, respectively. The mean duration of an ON-OFF cycle is thus 1 second, and the mean duration of an execution is 125 seconds.

When a new interactive application has to be started, a script is ran to open a VM, execute our Java program within the VM to emulate an interactive execution of the OpenOffice Spreadsheet application, and then close the VM. One of the input parameter of the Java program is the name of a file, chosen randomly among 25 files that we generated. Each file gives the number of ON-OFF cycles, as well as the duration of each ON or OFF phase. Figure 6 gives an example of such a file in the case of 4 ON-OFF cycles.

Phase	Time (ms)
ON	153
OFF	36
ON	1986
OFF	43
ON	1612
OFF	561
ON	751
OFF	183

Fig. 6: Example of a file with 4 ON-OFF cycles.

E. Web Service: permanent application

We simulate a web server with a simple client/server program that uses socket communication. As well, multi-threading programming is used to process multiple client request simultaneously. Client request are generated according to a Poisson process. Once the server accepts a socket, a thread is spawned to handle it. The thread performs an iterative computation. An input parameter of the thread controls the number of iterations, and thus the duration of the execution. The parameters are given in such a way that processing times of requests have an exponential distribution with mean 5 seconds.

VI. Main Results

In this section, we present the validity of the mathematical models presented in Section III. For this, we shall measure the mean response time of the applications described in Section V and compare them with those predicted by the mathematical models. We shall measure the accuracy of the predication of the mathematical models using relative error, i.e.,

$$RelErr(\%) = 100 \, \frac{\|T_{meas} - T_{mod}\|}{T_{meas}} \tag{7}$$

where T_{meas} and T_{mod} is the measured mean response time and the theoretical mean response time, respectively.

A. Parameter estimation

In order to compute the theoretical mean response times, we first need to calibrate the model, that is to determine the values the mean service times, $1/\mu_i$s. For this purpose, we ran one instance of a VM in isolation and executed a process inside it. That is, we wrote a script that opened a VM, executed the process (Scilab or OpenOffice) and then closed the VM, and this was repeated several times. From these experiments, we computed the empirical mean service times for the opening and closing of a VM and the execution of different applications.

The findings of these experiments were:

[2]The Apache OpenOffice Software Development Kit is an add-on for the AOO office suite. It provides the necessary tools and documentation for programming the AOO APIs and creating own extensions (UNO components) for Apache OpenOffice.

- Virtual Machines : the mean time to open and close a VM, i.e., $\frac{1}{\mu_0}$ and $\frac{1}{\mu_{R+1}}$ were 65 and 11 seconds, respectively.

- CPU-intensive applications : the time of execution of the Scilab program in the VM was 122 seconds. Taking into account the values obtained above for opening and closin a VM, we can conclude that the mean response time of a CPU-intensive job executed in a VM is 198 seconds.

- Interactive applications : as already mentioned in Section V-D, the mean number of ON-OFF cycles is $\frac{1}{p} = 125$ ms, the mean duration of an ON phase is $\frac{1}{\mu_r} = 800$ ms, and the mean duration of an OFF phase is $\frac{1}{\beta_r} = 200$ ms. The mean processing time of the Openoffice application is thus 125 seconds. Taking into account the values obtained above for opening and closin a VM, we can conclude that the mean response time of an interactive job executed in a VM is 201 seconds.

- Permanent applications : as already mentioned in Section V-E, requests have an exponentially distributed service time with mean 5 seconds. We change the arrival rate of requests generated by the client in order to change the load of the server.

These results are summarized in Table I.

Parameter	Value
$1/\mu_0$	65 sec
$1/\mu_{R+1}$	11 sec
$1/\mu_r,\ r \in \mathcal{B}$	122 sec
$1/\mu_r,\ r \in \mathcal{I}$	0.8 sec
$1/\beta_r,\ r \in \mathcal{I}$	0.2 sec
$1/p_r,\ r \in \mathcal{I}$	125 cycles

TABLE I: Parameters of an execution in a VM

Now that we have computed the different parameters of the model, we proceed to the validity of the model. Since the average facilities utilization in data center is estimated to be not high (56 % according to a recent report by McKinsey [3]), we consider in the following different scenarios with only light and medium load.

B. Applications in isolation

In this section, we shall study the case where there is only one type of application (either CPU-intensive, or interactive, or permanent) in the system, i.e., $R = 1$. Note that the difference with the experiments in the section of parameter estimation is that in that section only one instance of the application was running at any given time whereas in the current set of experiments several instances of the applications could be running concurrently. The arrival epochs of instances were generated according to a Poisson process. In each of the experiments, 100 instances of the application were generated, and they were scheduled using cron. Each instance opened a VM, ran the application and then closed the VM.

[3]McKinsey analysis. Available: www.mckinsey.com

We compare the results for two different value of system load : $\rho_{cpu} = 0.2$ and $\rho_{cpu} = 0.5$. In table II the theoretical mean response time and the percentage relative error is given for the three types of applications. We observe that the model predicts quite well the measured mean response time with a relative error under 4%.

C. CPU-intensive and Interactive Applications

Next, we study the case when both interactive and CPU-intensive applications can run concurrently on the same CPU. For this scenario, we generated according to a Poisson process and scheduled with cron the arrival epochs of the first 100 jobs for each type of application. Each one of the executions opens a VM, executes the OpenOffice or the Scilab program and closes the VM when the program is finished.

The experiments were conducted for two different system loads : (i) load of each type equal to 0.1; and (ii) load of CPU-intensive applications equal to 0.2 and that of interactive applications equal to 0.3. In the first case, the total utilization rate is 20 %, while it is 50 % in the second case.

In table III we present the results of these experiments. We observe that the relative error increases with the total system load but this error remains under 4% for a system load of 50 %.

D. CPU-intensive and Permanent Applications

Finally, we study the case when both one permanent application and CPU-intensive jobs can run concurrently on the same CPU. The arrival rate of client requests on the server are generated in such a way that the QS assumption holds.

The experiments were conducted for two different system loads : (i) load of permanent application equals to 0.1 and load of CPU-intensive jobs equals to 0.4; and (ii) load of permanent application equals to 0.1 and load of CPU-intensive jobs equal to 0.6. In the first case, the total utilization rate is 50 %, while it is 70 % in the second case.

In table IV we present the results of these experiments. We observe that for a total load equals to 70%, the estimation error is below 6 % and 5 % for CPU intense application and for the permanent jobs, respectively. When the total load is 50 % we obtain a relative error of 8.8 % for the CPU-intensive applications and less than 7 % for the permanent jobs.

VII. Conclusion

We have proposed simple performance models for applications executed in a cloud computing platform under the SaaS model. These models are applicable when each application instance is executed whitin a virtual machine running on a computing node of a data-center. Our main contribution is

Load	CPU Intensive		Interactive		Permanent	
	Model Mean Time	Relative Error	Model Mean Time	Relative Error	Model Mean time	Relative Error
0.2	247 sec	1.2%	244 sec	1.21%	6.25 sec	3.2%
0.5	396 sec	2.2%	376 sec	2.58%	10.0 sec	3.9%

TABLE II: Predicted mean response time and its relative error for CPU-intensive, Interactive and permanent applications under different loads.

Load CPU-intensive	Load Interact.	Total Load	Model Mean Time	Relative Error
0.1	0.1	0.2	246 sec	1.99%
0.2	0.3	0.5	383 sec	3.03%

TABLE III: Comparison of measured and predicted mean times when interactive and CPU-intensive applications are executed concurrently.

Load CPU-intensive	Load Permanent	Total Load.	Model Mean Time CPU-intensive	Relative Error	Model Mean Time Permanent	Relative Error
0.4	0.1	0.5	386.1 sec	8.8%	12.5 sec	6.4%
0.6	0.1	0.7	551.5 sec	5.53%	26.01 sec	4.9%

TABLE IV: Comparison of measured and predicted mean times when permanent and CPU-intensive applications are executed concurrently.

to explicitly take into account the different behaviors of the different classes of applications (interactive, CPU-intensive or permanent applications), and to propose simple queueing formulae to evaluate their performances. Although simple, these formulae are fairly accurate as shown by the experiments done on a real virtualized platform. Future work includes the generalization of our results to the case of several permanent applications executed on the same node. We shall also investigate the impact of machine failures and migration of jobs on the response times. Another direction is to compute optimal load-balancing policies using the results on the mean response time on a single CPU.

Acknowledgment

This work has been partially supported by grant ANR-11-INFR-001. The authors would like to thank Matthieu Jonckheere and Josselin Vallet for helpful discussions.

References

[1] P. Mell and T. Grance, "The nist definition of cloud computing (draft)," *NIST special publication*, vol. 800, no. 145, p. 7, 2011.

[2] M. Mahjoub, A. Mdhaffar, R. Halima, and M. Jmaiel, "A comparative study of the current cloud computing technologies and offers," in *Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on*, 2011, pp. 131–134.

[3] M. Turner, D. Budgen, and P. Brereton, "Turning software into a service." *Computer.*, vol. 36, no. 10, pp. 38–44, 2003.

[4] "Xen." [Online]. Available: http://www.xenproject.org/

[5] "Kvm." [Online]. Available: http://www.linux-kvm.org/

[6] R. Lent, "Evaluating the performance and power consumption of systems with virtual machines," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, 2011, pp. 778–783.

[7] D. A. Menascé, "Virtualization: Concepts, applications, and performance modeling," in *The Computer Measurement Groups'2005 International Conference*, Orlando, FL, USA, 2005.

[8] F. Benevenuto, C. Fernandes, M. Santos, V. Almeida, J. Almeida, G. Janakiraman, and J. Santos, "Performance models for virtualized applications," in *ISPA 2006 Workshops*, ser. Lecture Notes in Computer Science, G. Min, B. Martino, L. Yang, M. Guo, and G. Rünger, Eds., vol. 4331. Springer Berlin Heidelberg, 2006, pp. 427–439. [Online]. Available: http://dx.doi.org/10.1007/11942634_45

[9] J. Anselmi and I. Verloop, "Energy-aware capacity scaling in virtualized environments with performance guarantees," *Perform. Eval.*, vol. 68, no. 11, pp. 1207–1221, Nov. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.peva.2011.07.004

[10] F. Baccelli, F. I. Karpelevich, M. Y. Kelbert, A. N. Puhalskii, A. A.and Rybko, and Y. M. Suhov, "A mean-field limit for a class of queueing networks," *Journal of Statistical Physics*, vol. 66, no. 3-4, 1992.

[11] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang, "Energy-aware autonomic resource allocation in multitier virtualized environments," *IEEE Trans. on Services Computing*, vol. 5, no. 1, January-march 2012.

[12] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. Kozuch, "Optimality analysis of energy-performance trade-off for server farm management," *Perform. Eval.*, vol. 67, no. 11, pp. 1155–1171, Nov. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.peva.2010.08.009

[13] M. M. and H. M., "A performance study on the vm startup time in the cloud," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 423–430.

[14] L. Kleinrock, "Time-shared systems: A theoretical treatment," *J. ACM*, vol. 14, no. 2, Apr. 1967.

[15] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *J. ACM*, vol. 22, no. 2, pp. 248–260, Apr. 1975. [Online]. Available: http://doi.acm.org/10.1145/321879.321887

[16] "Virtualbox." [Online]. Available: https://www.virtualbox.org/

[17] "Cron." [Online]. Available: http://www.linuxmanpages.com/man5/crontab.5.php

[18] "Scilab." [Online]. Available: http://www.scilab.org/

[19] M. D. Lara, 2005. [Online]. Available: AvailableOnline:http://www.ing-mat.udec.cl/renewres05/materials/lecture_english/TPs/tp6/TP6b/english_Plant_alloc_II.html

[20] "Openoffice." [Online]. Available: http://www.openoffice.org/