ESAIM: PROCEEDINGS, December 2013, Vol. 43, p. 136-146
S. Descombes, B. Dussoubs, S. Faure, L. Gouarin, V. Louvet, M. Massot, V. Miele, Editors

MACROSCOPIC MODELIZATION OF THE CLOUD ELASTICITY *

J.-M. Etancelin[1], S. Faure[2], T. Fevrier[3] et C. Huynh[4]

Résumé. La gestion des ressources informatiques à la demande est une caractéristique propre à l'informatique en nuage. Pour que cela devienne une réalité, les fournisseurs de solutions en nuage ont besoin d'un modèle mathématique leur permettant de simuler l'allumage et l'extinction des ressources en fonction de la demande. Cet article propose une première étape de modélisation basée sur des outils issus de la simulation d'écoulements de fluides. D'une part, nous traiterons la question du temps de service dans l'optique de satisfaire simultanément les contraintes du fournisseur et du client. D'autre part, nous complèterons le modèle afin de simuler un comportement élastique des ressources du nuage. Ce comportement sera établi en temps réel et en fonction des demandes éventuellement concurrentes des clients. Nous intégrerons également au modèle des mécanismes utilisés en pratique dans les serveurs comme l'arrêt de requêtes trop longues. Ces problématiques seront illustrées par des simulations numériques menées à l'aide d'un schéma volume fini implicite.

Abstract. In order to achieve its promise of providing information technologies (IT) on demand, cloud computing needs to rely on a mathematical model capable of directing IT on and off according to a demand pattern to provide a true elasticity. This article provides a first method to reach this goal using a "fluid type" partial differential equations model. On the one hand it examines the question of service time optimization for the simultaneous satisfaction of the cloud consumer and provider. On the other hand it tries to model a way to deliver resources according to the real time capacity of the cloud that depends on parameters such as burst requests and application timeouts. All these questions are illustrated via an implicit finite volume scheme.

Introduction

Is it necessary to define Cloud computing since it is heard and overheard? Cloud computing is about revolutionizing the Information Technology (IT) consumption behavior without questioning the underlying technologies, be it applications or infrastructure. Cloud computing is about doing more business with less i.e. fewer infrastructures (servers, network, and storage) and less effort (self service portal, orchestration).

It is important to notice that Cloud computing is enduring an inflection point; it's all about shared elastic capabilities [4]. Let's quote a Gartner's analysis that is often reflected in reality: *"Vendors across many*

* *This work was supported in part by AMIES (http://www.agence-maths-entreprises.fr) and by the GDR CNRS Mathématiques et Entreprises (http://www.maths-entreprises.fr)*

[1] Laboratoire Jean Kuntzmann, Université Joseph Fourier, Grenoble, France. Mail : Jean-Matthieu.Etancelin@imag.fr

[2] CNRS, Laboratoire de Mathématiques d'Orsay, Université Paris-Sud, Bâtiment 425, 91405 Orsay Cedex, France. Mail : sylvain.faure@math.u-psud.fr

[3] Laboratoire de Mathématiques d'Orsay, Université Paris-Sud, Bâtiment 425, 91405 Orsay Cedex, France. Mail : tony.fevrier@math.u-psud.fr

[4] Cisco Systems France, L'Atlantis, 11 rue Camille Desmoulins 92782 Issy les Moulineaux, France. Mail : chuhuynh@cisco.com

categories have grabbed the term and use it for marketing purposes. Hosting solutions that have a pay-per-user-per-month pricing model, but without shared elastic capabilities, are being called cloud. As cloud is used to apply to an ever-widening set of external services without regard to sharing, elasticity and the other characteristics that set cloud apart, the term becomes even more unclear. Companies may buy these cloud services and then be surprised to find that they do not get the agility and cost savings promised by cloud computing. This has pushed cloud toward the Trough of Disillusionment." [5]

A first solution is a deal where each cloud consumer subscribes for a given amount of resources (number of virtual machines, network, and storage) and has a means of manually tune up and down consumption according to spotted demands. Such solution assumes that cloud consumers know exactly and timely what their needs may be. Unfortunately, reality check tends to prove the opposite: cloud consumer candidates are used to own IT resources and ignore completely how they are really used.

Our challenge is to fill the gap with a quantitative analysis. What if IT resources can be provisioned and deprovisioned automatically using a small set of variables, such as Service Level Agreement (SLA), reserved resource, elasticity ratio, total resource pool size?

Unlike some research, not any assumption on application type is made. Our model is designed to be attractive to as many cloud providers as possible. There is no use of consumer behavior statistics, since our model has to be applicable to local cloud providers who do not have the same range of IT resources than Google, Amazon, and the likes.

SLA is narrowed to two values: upper and lower watermarks on the service time. It's tempting to reflect a more sophisticated model. However, the simpler is the model, the faster cloud provider will implement. The upper watermark is the maximum delay a cloud consumer may tolerate. Above this limit, a cloud consumer expects his counterpart to activate elasticity to absorb pressure. The lower watermark is the minimum delay a consumer is ready to pay for. Under this limit, a cloud consumer expects his counterpart to save costs by deprovisioning IT resources.

Reserved resource is an amount of IT elements that a cloud subscriber deems necessary to have permanently, no matter it is used or idle. Elasticity ratio is a number of IT resources that is provisioned and deprovisioned according to temporary needs. The total resource pool size is the total capacity set by a cloud provider. It is the sum of all reserved pools and the elasticity pool shared among all cloud consumers.

First, the model from Jaisson and de Vuyst is exposed and adjusted to the cloud context. Second, a mechanism is added to satisfy upper and lower service bounds. It consists in the optimization of a cost function. This allows us to handle concurrent requests in the cloud. A model considering the drop of requests when the server is saturated is also implemented. The last part is devoted to use our tool in order to attribute a first IT set to a consumer according to his forecast of consumption.

1. Adjustment of Jaisson-de Vuyst model to the cloud

1.1. **Description of the initial model**

The following model for multithreads systems has been developed by Jaisson and De Vuyst [3]. It mimics the fluid flow behavior in a macroscopic way. Indeed, microscopic models become irrelevant when big amounts of data have to be scheduled and the cloud purpose is to fill up the virtual machines (VM) constituting it of requests to treat a lot of data. Moreover, it introduces a space variable to take into account memory and delay features of the transport of information. In the following, we reuse Jaisson's and De Vuyst's notations, see [3].

Let us consider a system delivering services with a capacity ϕ_o corresponding to the maximal requests flux it can encounter. A variable $x \in]0,1[$ representing the rate of completion of a task in the system is introduced. Thus, one can define the space and time dependant density $\rho(x,t)$ of completed tasks at level x at any time t

and then the total mass $m(t)$ of requests in the system :

$$m(t) = \int_0^1 \rho(x,t)dx. \tag{1}$$

The input rate of requests is denoted by ϕ_i.

To derive the model equations, a mass balance is written. The variation of the current mass of requests being processed between two levels of completion w_l and w_r is the difference between the entering flux at w_l and the leaving flux at w_r:

$$\frac{d}{dt}\int_{w_l}^{w_r} \rho(x,t)dx = \frac{\rho(w_l,t)\phi_o}{m(t)} - \frac{\rho(w_r,t)\phi_o}{m(t)}. \tag{2}$$

Making w_r tend to w_l in (2) gives the following Partial Differential Equations (PDE):

$$\partial_t\rho + v(t)\partial_x\rho = 0, \quad x \in]0,1[, \quad t > 0,$$
$$v(t) = \frac{\phi_o}{m(t)}.$$

However this transport equation allows the velocity, $v(t)$, to tend to infinity when the mass goes to zero that is physically impossible. That's why a limiter is used to have a maximal velocity equal to the system capacity ϕ_o:

$$v(t) = \frac{\phi_o}{\max(1, m(t))}.$$

The integration of the transport equation on space gives a mass conservation Ordinary Differential Equation (ODE). To complete the model, initial conditions at $t = 0$ on the flux and the mass and a boundary condition at $x = 0$ (input flux) are added. It comes:

$$\partial_t\rho + v(t)\partial_x\rho = 0, \quad x \in]0,1[, \quad t > 0, \tag{3a}$$
$$\partial_t m = \phi_i(t) - \rho(1,t)v(t), \quad t > 0, \tag{3b}$$
$$v(t) = \frac{\phi_o}{\max(1, m(t))}, \tag{3c}$$
$$\rho(x,0) = \rho^0(x), \tag{3d}$$
$$m(0) = \int_0^1 \rho^0(x)dx \geq 0, \tag{3e}$$
$$\rho(0,t)v(t) = \phi_i(t). \tag{3f}$$

In the context of the cloud, the evaluation of the service time of the system is an important issue. We denote by $d(x, t_0, x_0)$ the time for a request to be completed at level x given that it was at level x_0 at time t_0. So the whole service time of a task at time t is defined by $D(t) = d(1, t, 0)$ and d is transported at the velocity $v(t)$:

$$\partial_t d + v(t)\partial_x d = 1, \quad x \in]0,1[, \quad t > 0, \tag{4a}$$
$$d(0,t,0) = 0. \tag{4b}$$

A change of variable $u = t - d$ in (4) gives the following homogeneous system over service time:

$$\partial_t u + v(t)\partial_x u = 0, \quad x \in]0,1[, \quad t > 0, \tag{5a}$$
$$u(0,t) = t, \tag{5b}$$
$$D(t) = t - u(1,t). \tag{5c}$$

The quantity of interest, $D(t)$, is computed by solving simultaneously the two systems of equations (3) and (5) with the numerical scheme detailed in section 1.3.

1.2. **Adjustment to the cloud**

The cloud is made of an initial set of virtual machines (VM) which can be allocated on demand to cloud consumers. Each customer subscribes to a number of VM for a certain duration, typically a year, usually three years. Resignation can happen at any time (its obtention is explained later in Section 2.4). This number corresponds to a certain capacity ϕ_o which belongs to the cloud consumer. Once assigned to a consumer, a VM is no longer available nor accessible to other clients. It is called multi-tenancy and implies that cloud consumers share the resources provided by the cloud provider with all the required confidentiality, integrity, and availability levels.

As a consequence, to each cloud user corresponds one system as above with its own capacity ϕ_o and its own set of equations to solve: systems (3) and (5). There are as many independent systems as cloud consumers, since there are no communication between them. There are also a pool of unassigned VM that a cloud user may request according to its SLA and cloud provider commitment to elasticity (see Section 2.2). Without any client, the pool may be the whole capacity of the cloud provider. As clients keep coming, the pool size decreases, unless the cloud provider refills it by adding extra capacity.

In Section 2.3 the model will be enriched in order to integrate the possibility for the VM to drop requests when the service time is too long. It reproduces the usual behavior of servers for timeout requests.

1.3. **Numerical scheme**

Following [2], the system (3)-(5) is solved by a Finite Volume method (see for example [1]). The domain $[0;1]$ is discretized by N control volumes defined by $[x_{j-\frac{1}{2}}; x_{j+\frac{1}{2}}]$ where $x_{j+\frac{1}{2}} = jh,\ j \in \{0,\dots,N\}$ and $h = 1/N$. The cells centers are the points $x_j = (j-1/2)\,h,\ j \in \{1,\dots,N\}$.

The time discretization is ensured by an upwind implicit scheme with a time step dt^n which is unconditionally stable and can be solved explicitly. Hence, for $j \in \{1,\dots,N\}$, the unknowns are meant to be approximations of the cell averages with $t^n = t^{n-1} + dt^n$ for $n = 1,\dots,N_t$ and $t^0 = 0$:

$$\rho_j^n = \frac{1}{h}\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} \rho(x,t^n)dx, \tag{6}$$

$$m_j^n = \frac{1}{h}\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} m(x,t^n)dx, \tag{7}$$

$$u_j^n = t^n - \frac{1}{h}\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} d(x,t^n)dx. \tag{8}$$

Using the above notations, the full discretization of the system (3)-(5) is given by the following scheme, with $\lambda^n = dt^n/h$:

$$v^n = \frac{\phi_o}{\max(1, m^n)}, \tag{9a}$$
$$\rho_1^{n+1} = \rho_1^n - \lambda^n(\rho_1^{n+1}v^n - \phi_i^{n+1}), \tag{9b}$$
$$\rho_j^{n+1} = \rho_j^n - \lambda^n(\rho_j^{n+1} - \rho_{j-1}^{n+1})v^n, \tag{9c}$$
$$m^{n+1} = m^n + dt^n(\phi_i^{n+1} - \rho_N^{n+1}v^n), \tag{9d}$$
$$u_1^{n+1} = t^{n+1}, \tag{9e}$$
$$u_j^{n+1} = u_j^n - \lambda^n(u_j^{n+1} - u_{j-1}^{n+1})v^n, \tag{9f}$$
$$D^{n+1} = t^{n+1} - u_N^{n+1}. \tag{9g}$$

1.4. **Numerical validation**

The model is validated choosing different input request rates. In the following, we consider a 24 hour simulation with a time discretization of 1 minute (i.e. 1,440 time-steps). For the spatial discretization, 1,000 grid points are considered. The cloud consumer is using a cloud resource capacity of 120 requests per time-step. On Figure 1, the three input rates considered present a plateau of height 120, 150 and 200 requests per time-steps between 6 and 16 hours. In the first case, the system can process the whole input at its maximum speed whereas, for the others, the system is saturated and the service time increases. One can note that the bigger the amount of saturating requests is, the longer the service time takes to fall back to decent.

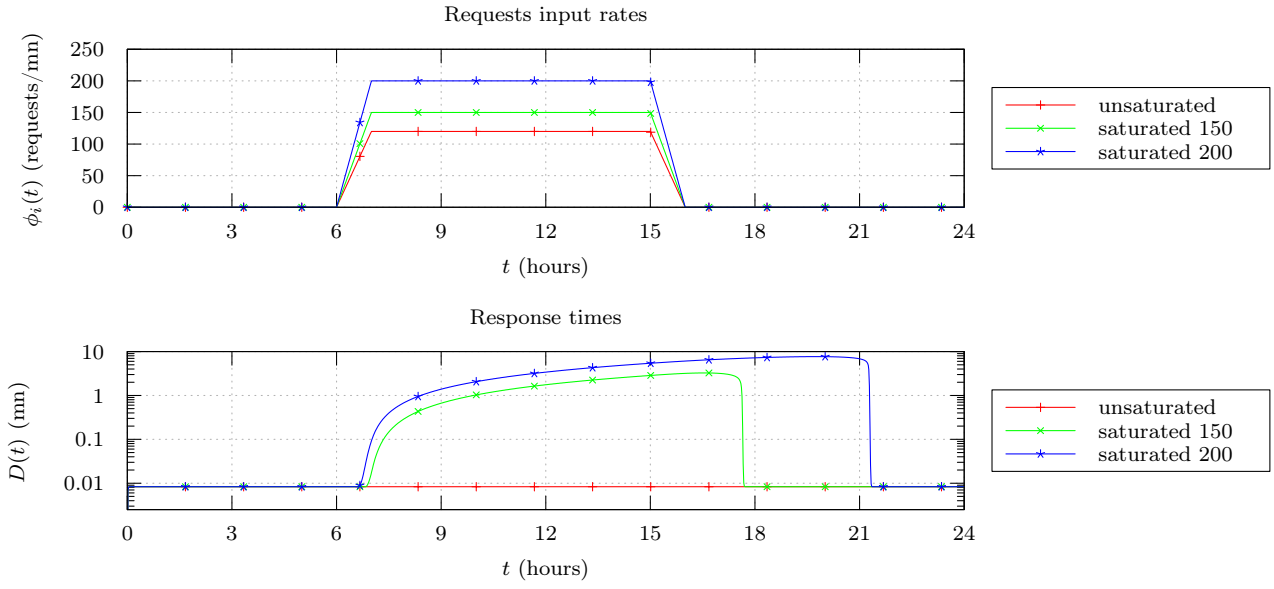

FIGURE 1. Basic model validation.

2. Service time optimization

2.1. The cost function

The easiest way to control the service time is to adjust the amount of resources allocated to a cloud consumer. In fact this time varies with the number of VMS and their computational characteristics. In our model, it means to vary ϕ_o.

As service time is a key quantity in networks, several optimizations can be done. From an end-user point of view, the service time should not be higher than a given time denoted T_{max}. For the cloud provider, it must be greater than T_{min} in order to prevent quality overachievement and keep resources available for incoming cloud consumers. These bounds over the service time depend on each kind of requests. For example, for a basic web service we could choose $T_{max} = 3$ seconds and $T_{min} = 0.1$ seconds. For data archiving, it could be respectively 6 hours and 1 hour.

In the real life, there is no means for a cloud provider to predict the requests input rate. The best that one can do is to monitor it. Therefore, optimizations are performed at a certain frequency, depending on the kind of service. Requests are supposedly independent, most of the time. Things may change when customer gets angry and retries the same request over and over again. We are interested in controlling the service time over small time windows of length Δt, regarding last input data available (such an assumption relates to the most of applications eligible to the cloud). We search for an optimal capacity $\phi_o^* \in]0; \phi_o]$ over Δt time windows with the minimization of the following cost function:

$$\mathcal{J}_n(\phi_o^*) = \int_{t^n}^{t^n+\Delta t} (D(t, \phi_o^*) - T_{min})^2 + (D(t, \phi_o^*) - T_{max})^2 dt. \tag{10}$$

We denote by ϕ_o^{*n} the capacity that realizes:

$$\mathcal{J}_n(\phi_o^{*n}) = \underset{\phi_o^* \in]0;\phi_o]}{\text{Min}} \mathcal{J}_n(\phi_o^*). \tag{11}$$

Therefore, the evolution of the optimal capacity for a cloud consumer is defined by a time dependent piecewise constant function $\phi_o^*(t)$:

$$\phi_o^*(t) = \phi_o^{*n} \quad \text{if } t \in]t^n; t^{n+1}].$$

However switching on or switching off VM takes a certain time. So modifying capacity is not instantaneous. That is the reason why we introduce a delay δt before applying the optimal capacity. This optimal capacity function becomes:

$$\phi_o^*(t) = \phi_o^{*n} \quad \text{if } t \in]t^n + \delta t; t^{n+1} + \delta t]. \tag{12}$$

The capacity optimization influence can be observed on Figure 2. The service time is computed for three cloud consumers with identical inputs. One of them has his capacity optimized to have $D(t) \in [0.5; 5]$. In the first case, the client is running a constant low capacity and sees his service time exploding at $t = 6$ hours. In the second case, he is using all his VM and then obtains a constant low service time except at $t = 15$ hours where his system is lightly saturated. Finally, in the last case, optimization is performed every 20 minutes, the cloud consumer uses the elasticity of the cloud. One can observe that the service time is correctly placed between the bounds, the capacity is reduced when the input is quite low and increased when it is necessary. We have therefore the evidence that there is no need to commit to a too low service time and both client and cloud provider are better off switching off VM when they don't use them.

2.2. Generalization to many cloud consumers

In a cloud, each consumer is allocated a set of VM of capacity ϕ_o he is the only one to use (part 1.2). So a part of size $\sum_{c \in \text{clients}} \phi_{o,c}$ of the cloud is fixed. If we denote by Φ the whole capacity of the cloud, $\Phi - \sum_{c \in \text{clients}} \phi_{o,c}$ is constitued of free VM. The aim of this paragraph is to determine a strategy to allocate these free VM to

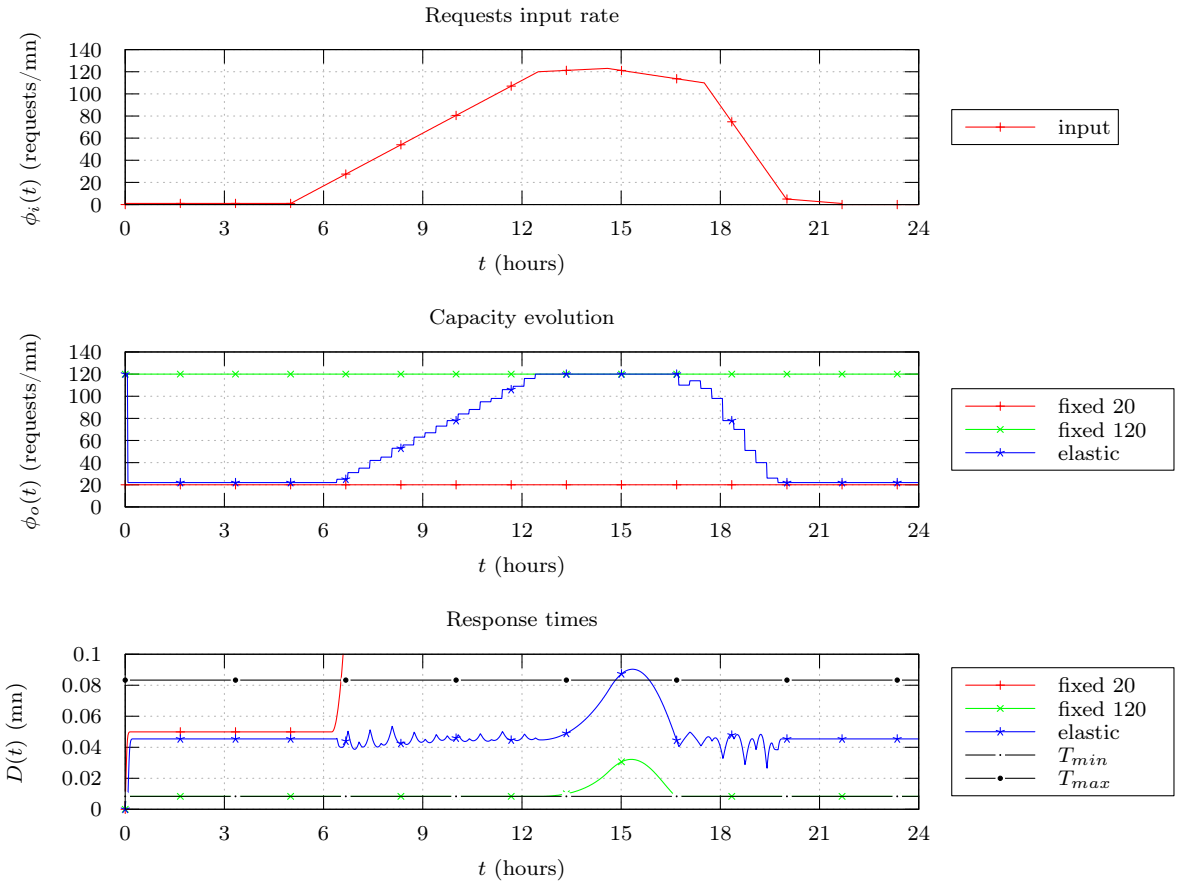

FIGURE 2. Capacity optimization influence.

cloud consumers according to the extra capacity they have bought and to the unassigned shared pool of the cloud.

Two things have to be defined: what part they can catch and when they need more capacity. For the first matter, each cloud user chooses and buys an option to use his extra capacity rate defined by $\phi_{eo} \in [0, 1]$. Thus if a cloud consumer has an allocated capacity of ϕ_o he can use a maximum of $\phi_o(1 + \phi_{eo})$. It is considered that he needs this extra capacity at time t^n if he is already using more than ϕ_o which means that $\phi_o^{*n} \geqslant \phi_o$.

Let t^n be the present time. Cloud consumers are ordered by decreasing service time. Thus the most saturated are served in priority. If a client uses less than ϕ_o (ie if $\phi_o^{*n} < \phi_o$), he does not need any extra capacity. So $\mathcal{J}$ is minimized on $]0; \phi_o]$. Otherwise (i.e. $\phi_o^{*n} \geqslant \phi_o$) he is allowed to use extra-resources. Nevertheless, the cloud provider can not always deliver him the whole ϕ_{eo}. Indeed, he can not provide more than the remaining free resources in the cloud. As in section 2.1, one can define the available capacity by a piecewise function:

$$\phi_{\text{av}}^n = \Phi - \sum_{c \in clients} \phi_{o,c}^{*n}. \tag{13}$$

So if $\phi_{\text{av}}^n = 0$, $\mathcal{J}$ is optimized on $]0; \max(\phi_o, \phi_o^{*n})]$ (the cloud consumer can keep the extra capacity he has already used). Else we calculate the rest of capacity he can take to satisfy his ϕ_{eo}: $\phi_o(1 + \phi_{eo}) - \phi_o^{*n}$. If

$\phi_o(1+\phi_{eo}) - \phi_o^{*n} \leq \phi_{\text{av}}^n$, $\mathcal{J}$ is optimized on $]0; \phi_o(1+\phi_{eo})]$, else it is optimized on $]0; \phi_o^{*n} + \phi_{\text{av}}^n]$.

An illustration of this algorithm is presented on Figure 3: there are two cloud consumers in the cloud. Each one owns a capacity of 120 and a free set a of capacity 12 is available. Both are authorized to take at most an extra capacity of 10% of their initial set (132). Clients don't have any reason to share the same numbers (120, 12, 10%) and the logic remains beyond 2 clients. Numbers are chosen for the sake of simplicity and clarity in the graphs. The same input (with a maximum of 132 requests per time-step) is launched by both, slightly before for the first client. As a consequence, he is the first to need an extra capacity: he takes all the available VM to satisfy his 10%. This relieves his service time which evolves in the expected interval. The second client becomes saturated later but no extra capacity is available until 16 hours. During this period, his service time grows and overpasses the upper bound. At 16 hours, the first consumer releases his extra capacity because of his input decrease. Thus the second one can take his 10% and stabilize his service time.

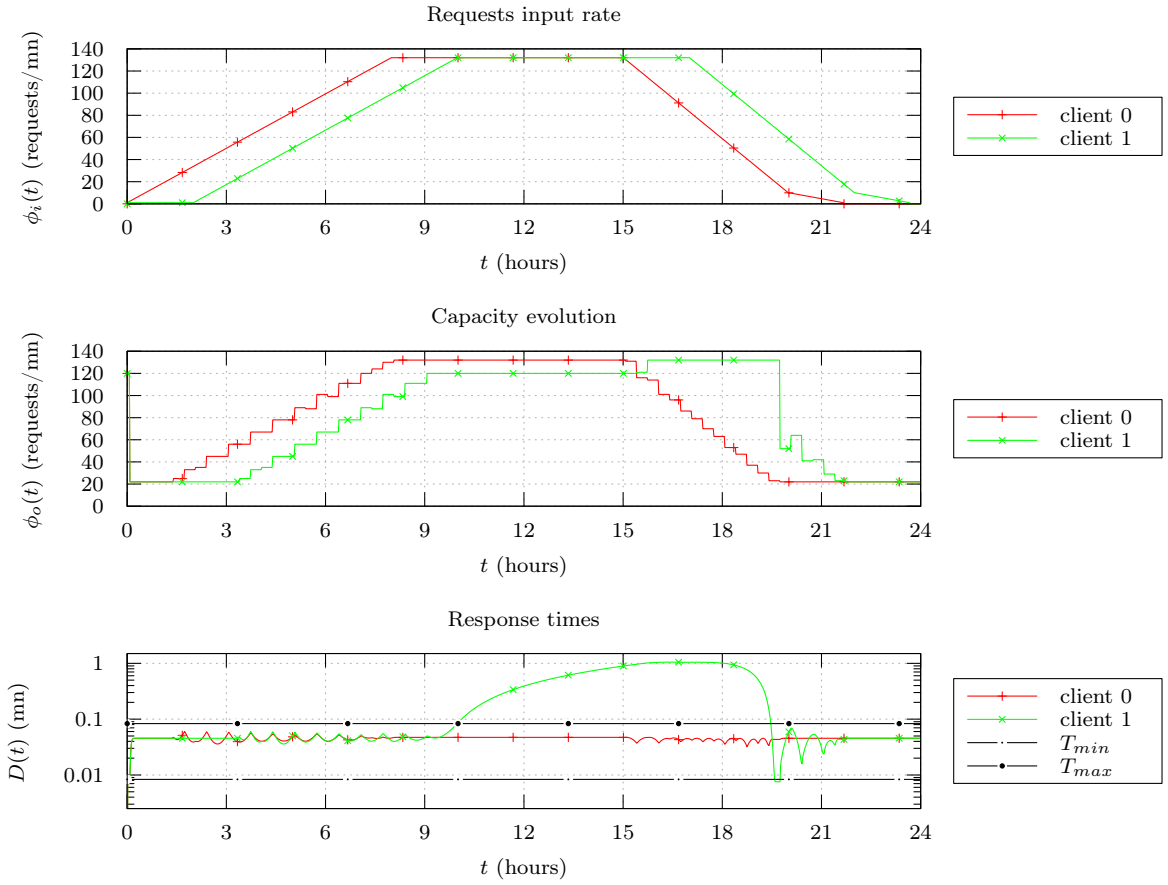

FIGURE 3. Two semi-shared ressources cloud consumers.

2.3. **Add request cessation in the model**

We also want to model the fact that web applications are typically designed to drop requests after a certain time delay. Such applications capture the end user behavior cancelling a request if an answer doesn't come back timely. Doing so, applications have the opportunity to catch up.

In this section, we integrate to our model a mechanism that lighten the servers workload: when the service time overpasses a certain threshold, the cloud must stop a chosen rate of requests.

Let's note $\tau(t)$ the damping of requests transport and T_{th} the threshold. It is assumed that the damping is uniform over the completion rate. Let $\tau(t)$ be equal to 0 when $D(t) \leq T_{th}$ and be a positive constant elsewhere. To model this suspension phenomenon, an additional term appears in the mass balance (2) representing the part of the deleted mass between w_l and w_r:

$$\frac{d}{dt}\int_{w_l}^{w_r} \rho(x,t)dx = \frac{\rho(w_l,t)\phi_o}{m(t)} - \frac{\rho(w_r,t)\phi_o}{m(t)} - \tau(t)\int_{w_l}^{w_r} \rho(x,t)dx.$$

The flux equation becomes:

$$\partial_t \rho + v(t)\partial_x \rho = -\tau(t)\rho(x,t), \quad x \in]0,1[, \quad t > 0, \tag{14}$$

$$v(t) = \frac{\phi_o}{\max(1, m(t))}. \tag{15}$$

$$\tag{16}$$

The requests density is still transported with the same velocity v but a damping factor when the service time is too high is added as a source term. Integrating this equation in space, the mass ODE reads

$$\partial_t m = \phi_i(t) - \rho(1,t)v(t) - \tau(t)m(t), \quad t > 0. \tag{17}$$

Regarding the numerical scheme, these new terms are treated implicitly.

We show the damping influence on Figure 4 where a cloud consumer is handled with different values for τ. The first case corresponds to previous results with no requests dropping. As one can see, the amount of input requests makes the client saturated between 12 and 19 hours. The use of requests cessation makes the service time below the threshold time T_{th} during the saturation period. It decreases as strong as τ is high. We notice also that as much as τ is increasing, the system needs less and less drops.

2.4. **Cloud consumer resources dimensioning**

When a consumer subscribes to the cloud, he chooses different types of applications. These applications can be translated in an equivalent number of requests per time-step and more precisely in a forecasting of input for a certain time sample. Note that the real input can be different from this forecasting. The client may not know exactly what resources amount he needs for his applications, so we could use the model as a sizing tool.

First we assume that there is one cloud consumer to size in the cloud. As it is explained above, with his subscription, the client provides an estimation of the future input. The idea is to launch the optimization algorithm on this forecasting input, the first ϕ_o being fixed to the total available cloud resources. The simulation gives a capacity profile from which we extract the maximum of capacity we propose to the client as the capacity he needs to handle his requests.

However if we want to size many cloud consumers, we can not give to everybody the cloud size to launch the first optimization. Indeed, nothing guarantees in this case that the sum of maximal capacities obtained remains in the limit of the cloud size. One solution is to size clients thanks to their estimation of input: each input mass is calculated (we integrate the input in time) and the part of the cloud received by the consumer is proportional to the mass he brings to the system. Then the system delivers to each cloud consumer the maximal capacity he could use for the real time input.

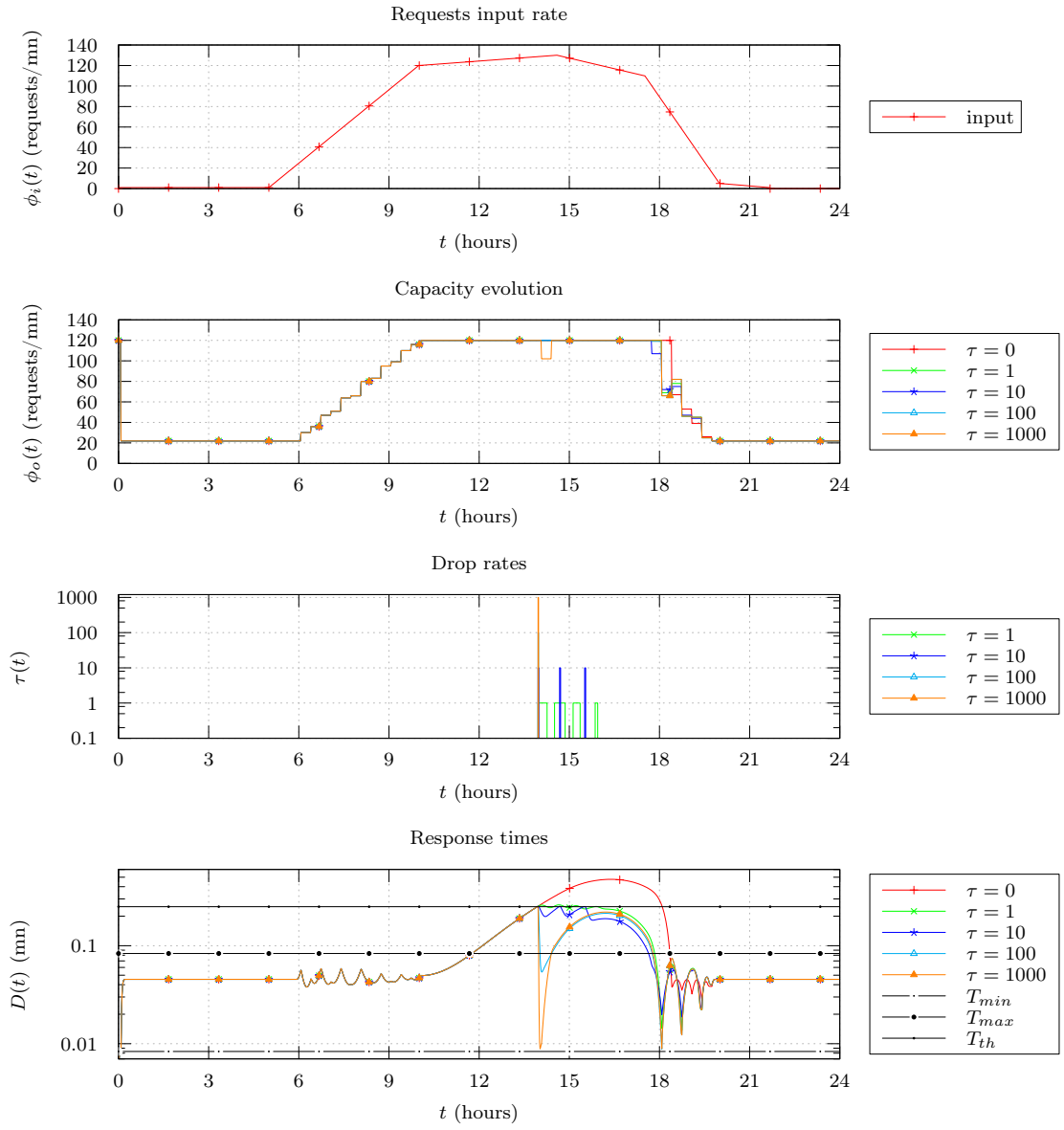

FIGURE 4. Influence of request cessation, $T_{th} = 3T_{max}$.

CONCLUSION

In this paper, the model from Jaisson and De Vuyst allowed us to make a first step in the modelling of the cloud elasticity. The way to modulate the IT resources was exposed: an optimization functional over the real time capacity was derived to satisfy not only the cloud consumer but also the cloud provider. Some validations tests were performed to show the usefulness of an "automatic" elasticity. The case of dropping requests was also considered when the system is saturated and we used our model to size a client (as a decision helper to a new client regarding an estimation of his input). Now, this work should be verified in a real case, and adjusted accordingly. In particular, the model could be adapted to get closer to the reality of cloud computing and

should be tested on real data sets. Some improvements could also be made with the addition of a variance term on capacity in the functional to prevent the excess of capacity oscillations.

References

[1] R. Eymard, T. Gallouët, and R. Herbin. Finite volume methods. In *Handbook of numerical analysis, Vol. VII*, Handb. Numer. Anal., VII, pages 713–1020. North-Holland, Amsterdam, 2000.

[2] P. Jaisson. *Systèmes complexes gouvernés par des flux: schémas de volumes finis hybrides et optimisation numérique*. PhD thesis, École Centrale Paris, 2006.

[3] P. Jaisson and F.D. Vuyst. An innovating pde model based on fluid flow paradigm for multithread systems. *Computer Networks*, 52(18):3318–3324, 2008.

[4] M. Kuperberg, N. Herbst, J.V. Kistowski, and R. Reussner. Defining and quantifying elasticity of resources in cloud computing and scalable platforms. *Karlsruhe Reports in Informatics*, 16, 2011.

[5] D. M. Smith. Hype cycle for cloud computing. G00230930, August 2012.