On the distribution of sequential jobs in random brokering for heterogeneous computational grids

Vandy Berten, Joël Goossens, Emmanuel Jeannot

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On the distribution of sequential jobs in random brokering for heterogeneous computational grids

Vandy BERTEN — Joël GOOSSENS — Emmanuel JEANNOT

N° 5499

Février 2005

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--5499--FR+ENG

On the distribution of sequential jobs in random brokering for heterogeneous computational grids

Vandy BERTEN* , Joël GOOSSENS† , Emmanuel JEANNOT‡

Thème NUM — Systèmes numériques
Projet Algorille

Rapport de recherche n° 5499 — Février 2005 — 23 pages

Abstract: This paper analyzes the way sequential jobs are distributed and the system behaves in a heterogeneous computational grid environment where the brokering is done in such a way that each Computing Element has a probability to be chosen proportional to its number of CPUs and, (new from the previous paper) its relative speed. We give the asymptotic behavior for several metrics (queue sizes, slowdown...) in several cases, or, in some case, an approximation of this behavior. We study those metrics in several workload specifications: for various loads (saturated or non saturated), with several distributions,... We compare our probabilistic analysis to simulation we performed, in order to validate our results.

Key-words: Grid brokering, multi-level scheduling, random brokering, stochastic workload, heterogeneous and distributed architecture.

* Vandy.Berten@ulb.ac.be - Research Fellow for FNRS-ULB (Fonds National de la Recherche Scientifique - Université Libre de Bruxelles, Belgium)
† Joel.Goossens@ulb.ac.be - ULB (Université Libre de Bruxelles, Belgium)
‡ Emmanuel.Jeannot@loria.fr - LORIA, Université Henri Poincaré, Nancy-1 (France)

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,

Distribution de jobs séquentiels dans le cas des grilles de calcul hétérogènes avec méta-scheduling aléatoire.

Résumé : Ce travail analyse la façon dont des jobs séquentiels sont distribués dans un environnement de grille de calcul où le brokering est fait en sorte que la probabilité qu'un site soit sélectionné pour l'exécution d'un job est proportionnelle à la puissance de ce site, c'est-à-dire le produit entre le nombre de processeurs et le facteur d'accélération de ces processeurs. Nous fournissons le comportement asymptotique d'un certain nombre de métriques, telles que la taille des files d'attente, le facteur de ralentissement (ou slowdown), ou le nombre de processeurs utilisés. Dans certains cas, nous proposons une approximation de ce comportement.

Mots-clés : meta-ordonnancement aléatoire, Grille, ordonnancement multi-niveau, charge stochastique, architecture hétérogène et distribuée

1 Introduction

Grid systems are the gathering of distributed and heterogeneous resources (CPU, disk, network, etc.). They are promising infrastructures for executing large scale applications or for providing computational power to everyone. In order to hide the complexity of grids to the users, executing environments (called middleware) are to be developed. A middleware aims at providing a programming API and an execution model for the applications. It relies on a set of services that enables the control of the resources, the deployment of the services, the execution of the applications, etc. The scheduling service has in charge the allocation of the tasks on the distributed resources. Therefore it is one of the key services for enabling performance and efficiency of the whole grid.

In this paper, we focus on a special kind of grid scheduler called a resource broker or a meta-scheduler. A meta-scheduler has in charge to dispatch client requests (jobs) onto computing elements (for instance a cluster) each of these being composed of several computing nodes (processors). Each computing element executes its own local scheduling policy for executing the subset of the jobs assigned to it. In general the local scheduler is called a batch scheduler. A common implementation of a batch scheduler consists in storing the jobs into a queue that follows a FIFO policy. Other techniques have been also studied; see for instance [7] for an overview.

In order to be efficient, a resource broker needs to know the duration of the submitted job on each node of the computing resources. This assumption is not always realistic, as the duration of the job might not be known before its completion. Indeed, this duration may depend on its parameters and data. It also requires the user to have run its job at least once and give obtained duration to the broker. In this work, we propose another approach where the duration of each job is not known but is given by a random variable with a fixed mean (for instance the duration follows an exponential law). Moreover, we do not assume that the submission of the jobs is known in advance. We suppose that the arrival date is also drawn with a random variable (for instance a Poisson law). This probabilistic approach is driven by the dynamic nature of grids where it is not always possible to precisely know in advance both duration and arrival date of jobs (as in a UNIX system). The proposed resource brokering algorithm is therefore a randomized one.

Resource brokering is an important subject since many middlewares use a meta-scheduler for allocating jobs. This is the case of EDG/EGEE that is one of the few production grid functioning 7 days per week, 24 hours per day. In this paper we study a randomized algorithm (random brokering) for allocating stochastic workload to a grid environment similar to the EDG/EGEE one. As in EGEE, we assume that computing elements of the target grid are heterogeneous. However, inside each computing elements, the processors are supposed homogeneous. Two cases are analyzed. In the first case, we assume that the grid is not saturated (it is submitted less jobs than it can execute). In the second case, the grid is saturated (it is submitted more jobs than it can execute). Each job is assumed to be sequential (requires only one CPU).

For each case, we compute the average queue length of each computing element (section 3). The queue length is an important metric since it helps to design and tune the local

batch scheduler. We also study the utilization of the grid: the number of CPU used on each computing elements (section 4). When, due to change of the load, the grid goes from the saturated case to the non-saturated case, we compute an approximation of the time required to empty the queues on each computing elements (section 5). Finally, jobs may be delayed depending on the load of the grid. We study this last metric (called the slowdown) for several distribution of the duration of the tasks (section 6). Each of these metrics is studied both analytically and with simulation experiments. Therefore, the contribution of this paper is an extensive comprehension of the behavior of random brokering on heterogeneous computational grids.

This paper generalizes the results presented in [1] which considers *homogeneous* grids (where CPUs have all the *same* speed). In this paper, we generalize those results by analyzing *heterogeneous* grids (where CPUs have *different* CPU speeds). We extend in this paper the properties considered in [1] by generalizing our proof techniques. We show in few cases, that the properties remain the same and consequently are not related to the CPU speeds (which is rather counter intuitive), and we give the more general result in other cases. We provide also in this paper analysis for additional distributions (e.g., uniform and erlang).

2 Background

2.1 Model of computation

In this paper, we will consider a quite simple but sufficiently realistic model of computational grid. The analysis we made is mainly focused on computational grids, as we assume that network latencies and transfer times are negligible and we thus do not take into account the data localization. In the grid we consider, there is a central *Resource Broker* (RB), to which each *Computing Element* (CE) is connected, and each client sends its jobs to that central RB. Without lack of generality, we use only one client, but several kinds of workloads have been simulated. Each CE $\mathbb{C}_i$ has a *relative speed* s_i, which is the relative speed of its CPUs compared to a reference CPU. That means that if we have two CPUs with relative speeds s_1 and s_2, a job which takes ℓ units of time on the first CPU to be processed would take $\ell \cdot \frac{s_1}{s_2}$ on the second one. We assume that our system is locally homogeneous, meaning that each CPU of a computing element has the same speed.

Each job j submitted on our system has mainly one parameter: a virtual length (or virtual execution time) j_ℓ, that is the time that would be taken by a reference CPU (with a relative speed equal to 1) for processing that job. On a CE with relative speed s, the job j will therefore use one CPU during j_ℓ/s units of time. j_ℓ/s is the effective length (or execution time) of j on a CE with relative speed s. We assume that, on one processor, we do not use parallelism or preemption (and then migration), and that our system is greedy[1].

[1] A system is said to be *greedy* (sometimes called *expedient*) if it never leaves any resource idle intentionally. If a system is greedy, a resource is idle only if there is no eligible job waiting for that resource.

2.2 Mathematical model

In the next sections, we will use the following notations: Our system is composed of $\mathcal{N}$ Computing Elements, called $\mathbb{C}_i (i \in [1 \dots \mathcal{N}])$. c_i refers to the number of CPUs of $\mathbb{C}_i$, and s_i to the relative speed of $\mathbb{C}_i$. $c_i \cdot s_i$ is the virtual number of CPUs on $\mathbb{C}_i$. $C \triangleq \sum_{i=1}^{\mathcal{N}} c_i \cdot s_i$ (the symbol $\triangleq$ means "is by definition" or "is defined as") is then the total number of virtual CPUs. We define the system load $\nu(t)$ as the total amount of virtual work received in $[0, t]$ divided by the product between the total number of virtual CPUs (C) and the total duration (t), or in other word, the total amount of (virtual) work received divided by the total amount of (virtual) work that the system could provide. Formally,

$$\nu(t) \triangleq \frac{\sum_{\{j \in \mathcal{J} | j_s \leq t\}} j_\ell}{C \cdot t}$$

where $\mathcal{J}$ is the set of jobs submitted to the system, and j_s is the job j's submission time. The interval $[0, t]$ is called the *observation period*.

We introduce here a notation: $f_1(t) \underset{t}{\sim} f_2(t)$ means that $\lim_{t \to \infty} \frac{f_1(t)}{f_2(t)} = 1$. Informally, we say that "$f_1(t)$ behaves asymptotically like $f_2(t)$". Notice that $f_1(t) \underset{t}{\sim} f_2(t) \Leftrightarrow f_1(t) = f_2(t) + \varepsilon(t)$, with $\lim_{t \to \infty} \frac{\varepsilon(t)}{f_2(t)} = 0$.

We assume that the arrival of jobs is a random process with an average delay λ^{-1} between two successive arrivals (e.g., the arrivals could be a Poisson process with rate λ, that is, the delay between two successive arrivals follows an Exponential law with the same rate of change). We also assume that the average virtual execution time ($\mathbb{E}[j_\ell]$) has a distribution with mean μ^{-1} (for instance an Exponential distribution with rate μ). We assume that jobs are independent (i.e., they do not share common ressources except CPUs, and there is no precedence constraints between jobs).

2.3 Random Brokering

2.3.1 Dispatching the jobs

In this work, we will study a particular case of brokering, where jobs are randomly dispatched, but with a distribution of probability based on the capacity of the different CEs.

In a EDG system, this corresponds to the case where the *rank*, in *fuzzy* mode, is `TotalCPUs * AverageSI00` (or `GlueHostBenchmarkSI00 * CEInfoTotalCPUs` in the glue schema).

The system we analyze is modelized as follows. Each job requires only one CPU, and the broker chooses one among all Computing Elements in such way that each $\mathbb{C}_i$ has a probability equal to $\frac{c_i \cdot s_i}{C}$ to be chosen.

It is easy to see (see for instance [1]) that, if inter-arrivals are independent, for each $\mathbb{C}_i$, the inter-arrival mean will tend towards λ_i^{-1}, where $\lambda_i \triangleq \lambda \frac{c_i \cdot s_i}{C}$, and the average execution time will be $\mu_i^{-1} \triangleq (\mu \cdot s_i)^{-1}$.

2.3.2 System load

We define here ν_i as being $\frac{\lambda_i}{\mu_i c_i}$. We have then $\nu_i = \frac{\lambda \frac{c_i \cdot s_i}{C}}{\mu s_i c_i} = \frac{\lambda}{\mu C} \triangleq \nu$.

It is not difficult to see that, in the average, $\nu(t) \underset{t}{\sim} \nu$; indeed (by the Law of Large Numbers),

$$\begin{array}{lll} \nu(t) & \triangleq \frac{\sum_{\{j \in \mathcal{J} | j_s \leq t\}} j_\ell}{C \cdot t} & = \frac{\|\{j \in \mathcal{J} | j_s \leq t\}\| \frac{\sum_{\{j \in \mathcal{J} | j_s \leq t\}} j_\ell}{\|\{j \in \mathcal{J} | j_s \leq t\}\|}}{C \cdot t} \\ & \underset{t}{\sim} \frac{\|\{j \in \mathcal{J} | j_s \leq t\}\|}{t} \frac{\mu^{-1}}{C} & \underset{t}{\sim} \frac{\mu^{-1}}{C \cdot \lambda^{-1}} = \nu \end{array}$$

ν will then be called the system load. If $\nu < 1$, the system will be said *non-saturated*, and if $\nu > 1$, the system will be considered as *saturated*. The case $\nu = 1$ will not be analyzed in this paper.

In the next two sections, we will first look at the queue size. We then shortly take a glance at two other metrics (number of used CPUs and resorption time), and finally, we analyze the average slowdown. From a theoretical point of view, the pure saturated case ($\nu > 1$) is not really interesting, because queues grow indefinitely and therefore the system "explodes". However, in real systems, the load is often time dependent, and an alternation of saturated and non-saturated phases can be observed. We are then interested by how queues are growing during a saturated phase, and how queues are decreasing when the load goes down to a non-saturated phase.

Next sections require a different approach if $\nu < 1$ and if $\nu > 1$ (i.e., if the system is saturated). They will therefore be splitted in two parts.

3 Queue size

As a first metric, we will here analyze the average queue size of computing elements. This metric is usefull, for instance for tuning the local scheduler, or for dimensioning the memory needs for buffers.

This study will be divided in two parts: the first one analyzing non saturated systems ($\nu < 1$), the second one saturated systems ($\nu > 1$).

3.1 $\nu < 1$

Let us first have a look at the case $\nu < 1$. We focus here on a single (arbitrary) $\mathbb{C}_i$, where the arrival is a Poisson process with rate λ_i and the execution time has an Exponential distribution with mean ${\mu_i}^{-1}$. Such a system is well known, and has been abundantly studied in the literature: this is a $M/M/c_i$ queuing system.

Let J_i be the number of jobs in $\mathbb{C}_i$ (running and waiting jobs), and $\mathbb{P}[J_i = n]$ the probability that the number of jobs in $\mathbb{C}_i$ is ni; we know (see for instance [9], page 371, section 8.5.2) that

$$\mathbb{P}[J_i = n] \underset{t}{\sim} b \begin{cases} \frac{(\nu c_i)^n}{n!} & \text{if } n \in [0, c_i] \\ \frac{\nu^n \cdot {c_i}^{c_i}}{c_i!} & \text{otherwise} \end{cases} \tag{1}$$

where

$$b \triangleq \left[\sum_{k=0}^{c_i - 1} \frac{(\nu c_i)^k}{k!} + \frac{(\nu c_i)^{c_i}}{c_i!} \frac{1}{1-\nu} \right]^{-1}$$

If Q_i is the asymptotic queue size, we have

$$\mathbb{P}[Q_i = n] = \begin{cases} \mathbb{P}[J_i = n + c_i] & \text{if } n > 0 \\ \\ \sum_{k=0}^{c_i} \mathbb{P}[J_i = k] & \text{if } n = 0 \end{cases}$$

Indeed, if there are $n > 0$ jobs in the queue, that means that those n jobs cannot be processed currently, because all CPUs are busy. There is therefore at that time n (in the queue) + c_i (currently processed) jobs on $\mathbb{C}_i$. If there are no jobs in the queue, the probability of this event is equal to $\mathbb{P}[J_i \leq c_i]$.

Hence,

$$\mathbb{P}[Q_i = n] \underset{t}{\sim} \begin{cases} b \frac{\nu^{n+c_i} \cdot {c_i}^{c_i}}{c_i!} & \text{if } n > 0 \\ \sum_{k=0}^{c_i} b \frac{(\nu c_i)^k}{k!} & \text{if } n = 0 \end{cases}$$

It is now easy to compute the average queue size:

Theorem 3.1 *If $\nu < 1$, the average queue size of $\mathbb{C}_i$ is*

$$\mathbb{E}[Q_i] \underset{t}{\sim} b \frac{\nu^{c_i+1} \cdot c_i^{c_i}}{c_i!(1-\nu)^2}$$

where

$$b \triangleq \left[\sum_{k=0}^{c_i - 1} \frac{(\nu c_i)^k}{k!} + \frac{(\nu c_i)^{c_i}}{c_i!} \frac{1}{1-\nu} \right]^{-1}$$

Proof :

$$\begin{aligned} \mathbb{E}[Q_i] &\underset{t}{\sim} \sum_{n=1}^{\infty} n \cdot b \frac{\nu^{n+c_i} \cdot {c_i}^{c_i}}{c_i!} \qquad \text{by definition of } \mathbb{E} \\ &= b \frac{(\nu c_i)^{c_i}}{c_i!} \sum_{n=1}^{\infty} n \nu^n \\ &= b \frac{(\nu c_i)^{c_i}}{c_i!} \frac{\nu}{(1-\nu)^2} \end{aligned}$$

Then,

$$\mathbb{E}[Q_i] \underset{t}{\sim} b \frac{\nu^{c_i+1} \cdot c_i^{c_i}}{c_i!(1-\nu)^2}$$

with the same b as above. □

Let us remark that the relative speed s_i of $\mathbb{C}_i$ does not appear in $\mathbb{E}[Q_i]$ when $\nu < 1$. The average queue is then "relative speed independent" in the non-saturated case.

3.2 $\nu > 1$

When $\nu > 1$, we can free the constraint that we are only looking at poissonnian systems. The only thing we need is that inter-arrivals and execution times have a finite average.

We are interested in the $\mathbb{C}_i$ queue size. Obviously, we cannot use the same technique as above, because asymptotically, the queue of a saturated CE is always infinite, regardless of the load provided that $\nu > 1$. We will focus on the average queue size at time t, that is, if $P_n(t)$ is the probability that there are n jobs in the queue at time t, we will consider $\sum_{n=0}^{\infty} nP_n(t)$.

Before estimating the queue size, we will need to know the number of arrived jobs before time t, as well as the number of jobs having left the system by time t.

3.2.1 Arrivals

Let $A_i(t)$ be the random variable which gives the number of jobs submitted between 0 and t (included) to $\mathbb{C}_i$, and $D_i(t)$ the random variable which gives the number of jobs having left $\mathbb{C}_i$ up to t. We have:

Lemma 3.1 *If the arrival process has λ_i^{-1} as inter-arrival mean, and a finite variance, then*

$$\mathbb{E}[A_i(t)] \underset{t}{\sim} \lambda_i t$$

Proof : It is known (see [4], page 372, chapter XI, section 6) that the number of jobs arrived between 0 and t tends asymptotically (in terms of t) towards a Gaussian with mean $\lambda_i t$ and variance $tv\lambda_i^3$, where v is the variance of inter-arrivals, which has to be finite. This is an extension of the central limit theorem. We have therefore that $\mathbb{E}[A_i(t)]$ behaves asymptotically as $\lambda_i t$ □

However, in practice, such a process converges quite quickly towards a Gaussian; a few dozen iterations are generally sufficient to have a really good approximation, for most distributions encountered in the kind of system we look at.

3.2.2 Departures

We will now show that the asymptotic behavior of $\mathbb{E}[D_i(t)]$ on t, when $\nu > 1$, is equal to $\mu_i t c_i$. We first need to remember a result from queuing theory: if $\nu > 1$ in a $G/G/c$ queue, the average time for having back an empty queue from a point where the queue is not empty is infinite. Moreover, the average length of an idle period[2] is finite. It is due to the fact that

[2] An idle period on a CE is a period during which at least one CPU is not running a job and which cannot be extended without containing a period where each CPU is busy. On a single processor, an idle period is a period during which the CPU is not running a job and that cannot be extended.

the state "empty queue" is transient; the number of visits to such a state is then (almost surely) finite, and the number of busy periods (between two successive idle periods) is finite. Therefore, the average length of a busy period is infinite, and the average length of an idle period is finite.

We have from this that if $I(t)$ is the sum of each idle period duration on $[0,t]$, then $(t - I(t)) \underset{t}{\sim} t$.

Let $D_i^k(t)$ be the number of jobs sent on $\mathbb{C}_i$, processed by the CPU k, and having left the system by t, ${D^*}_i^k(t)$ be the number of departures before t in a system where we compact the time line in order to remove every idle period, and $\delta_k(t)$ be the total idle time on CPU k between 0 and t. We have :

Lemma 3.2

$$D_i^k(t) = {D^*}_i^k(t - \delta_k(t))$$

and

$$t - \delta_k(t) \underset{t}{\sim} t$$

Proof : We will bring some changes to our system, in order to remove every idle periods. By definition of $D_i^k(t)$, we have that $D_i(t) = \sum_{k=1}^{c_i} D_i^k(t)$, and then

$$\mathbb{E}[D_i(t)] = \mathbb{E}[\sum_{k=1}^{c_i} D_i^k(t)] = \sum_{k=1}^{c_i} \mathbb{E}[D_i^k(t)] \tag{2}$$

Let us now look at one specific processor (let say k). If we observe the usage of that CPU, we can see a succession of idle and busy periods. Let i_1 be the first idle period (if such a period exists), and let d_1 be its duration. We can see that if we moved down by a time d_1 the arrival time of each job submitted after i_1 on this CPU, i_1 disappears, but we have that $D_i^k(t) = {D'}_i^k(t - d_1)$, where D' is the number of jobs leaving the "new" system. If we continue the same process until we have removed all idle periods, we will have that

$$D_i^k(t) = {D^*}_i^k(t - \sum_{\ell} d_\ell)$$

where there are no more idle periods in the system described by D^*, and, by definition, $\delta_k(t) = \sum_\ell d_\ell$. Clearly, $\delta_k(t) \leq I(t)$ (each idle period of the k^{th} CPU is counted in $I(t)$, but some idle periods from other CPUs are in $I(t)$ as well), and then $t - \delta_k(t) \underset{t}{\sim} t$. □

We can now show another lemma:

Lemma 3.3

$$\mathbb{E}[D_i^k(t) | \textit{no idle period in } [0,t]] \underset{t}{\sim} \mu_i t$$

Proof : If we do not have idle periods in $[0,t]$, the "departure events" is a stochastic process, where inter-arrival events are distributed as effective job lengths. For instance, if the job lengths distribution is exponential, the departures will be distributed as a Poisson process. We can then use the same reasoning as for Lemma 3.1, using ${\mu_i}^{-1}$ as inter-arrival mean. □

We now have what we need for estimating the number of departure before t:

Lemma 3.4

$$\mathbb{E}[D_i(t)] \underset{t}{\sim} \mu_i t c_i$$

Proof :

$$\begin{array}{lll}
\mathbb{E}[D_i(t)] & = \sum_{k=1}^{c_i} \mathbb{E}[D_i^k(t)] & \text{by Equation 2} \\
& = \sum_{k=1}^{c_i} \mathbb{E}[D^{*k}_i(t-\delta_k(t))] & \text{by Lemma 3.2} \\
& \underset{t}{\sim} \sum_{k=1}^{c_i} \mu_i(t-\delta_k(t)) & \text{by Lemma 3.3} \\
& \underset{t}{\sim} \sum_{k=1}^{c_i} \mu_i t & \text{because } t-\delta_k(t) \underset{t}{\sim} t \text{ (Lemma 3.2)} \\
& = \mu_i t c_i &
\end{array}$$

□

3.2.3 Queue size

Now we have the asymptotic behavior of A_i and D_i; we can therefore find the asymptotic behavior of Q_i.

Theorem 3.2 *If $\nu > 1$, we have*

$$\mathbb{E}[Q_i(t)] \underset{t}{\sim} \lambda_i t \left(\frac{\nu-1}{\nu}\right)$$

Proof : By definition of J ($J_i(t)$ is the number of jobs in the system – running and waiting – at time t), A and D, we have

$$\begin{array}{lll}
J_i(t) & = A_i(t) - D_i(t) & \\
\mathbb{E}[J_i(t)] & = \mathbb{E}[A_i(t)] - \mathbb{E}[D_i(t)] & \\
& \underset{t}{\sim} \lambda_i t - \mu_i t c_i & \text{By Lemma 3.1 and 3.4} \\
& = t(\lambda_i - \mu_i c_i) &
\end{array}$$

Of course, the queue size $Q_i(t)$ is equal to $\max(J_i(t) - c_i, 0)$, and therefore, in the case we are looking at (saturated and greedy system), we have almost always $Q_i(t) = J_i(t) - c_i$. Therefore

$$\begin{array}{lll}
\mathbb{E}[Q_i(t)] & \underset{t}{\sim} t(\lambda_i - \mu_i c_i) - c_i & \\
& \underset{t}{\sim} t(\lambda_i - \mu_i c_i) & \text{Because } c_i \text{ is constant regarding to } t \\
& = \lambda_i t \frac{\nu-1}{\nu} &
\end{array}$$

□

The slope of the queue growth is therefore $(\lambda_i - \mu_i c_i)$.

We can now calculate easily the total amount of jobs being in a queue of the whole system; we just need to sum up those queue sizes:

$$\begin{array}{rcl}
\mathbb{E}[Q(t)] & = & \sum_i \lambda_i t \frac{\nu-1}{\nu} \\
& = & \lambda t \frac{\nu-1}{\nu}
\end{array}$$

3.3 Experimental results

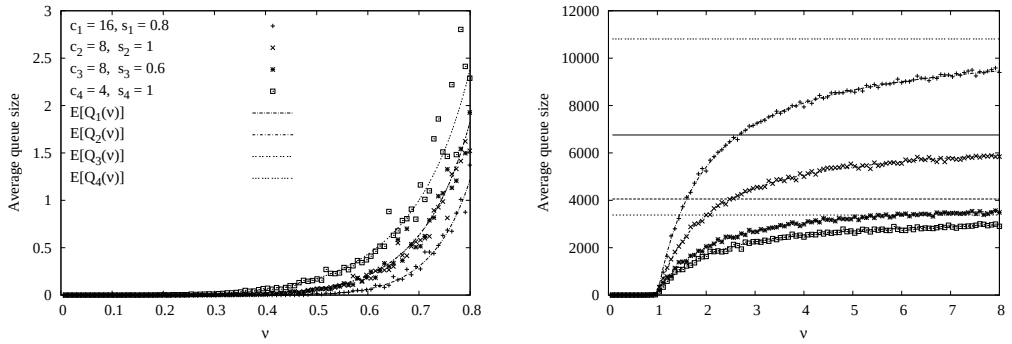

Figure 1: Queue Size observed in simulation and predicted, for non saturated system (left - Theorem 3.1) and saturated systems (right - Theorem 3.2). On the right plot, asymptotes are shown

In this paper, we are interested in various metrics, like the (average) queue size, the (average) slowdown, the (average) number of used CPUs... and in particular in their asymptotic behavior for various system loads. In order to illustrate that, we will plot those metrics (y-axis), observed by simulation or predicted by theoretical analysis, obtained for a (large) fixed time t, in function of the system load ν (x-axis). For instance, in Figure 1, we performed about one hundred of simulations, for various ν's going from 0 to 8 (0 to 0.8 for the left plot), and noticed the average queue size after a "long" constant time for each computing element (four dots vertically aligned); we also plotted in the same figure the values predicted by the theory (lines). The size (number of CPUs and the relative speeds) of the simulated CE is shown in the legend.

In the examples we picked from our simulations, we used the following parameters: the simulation duration (t) was 100 000 and the average inter-arrival delay (λ^{-1}) was 2. The four CEs had 16, 8, 8 and 4 CPUs (c_i), and the relative speeds were, respectively, 0.8, 1, 0.6, 1.

In the first plot (Figure 1, right side), we put our results for $\nu < 1$ (using Theorem 3.1, which is only valid for exponential distributions), and in the second plot, we show the case $\nu > 1$ (using Theorem 3.2). We did not put these two parts on the same plot for a simple reason of readability; the order of magnitude for the first part is around 10 (queue sizes are lower that 10 up to $\nu \simeq 0.98$), and around 10000 for the second part (the biggest queue size tends asymptotically towards ~ 10810).

We observe that there is an "inversion" around $\nu = 1$: when $\nu < 1$, $\mathbb{E}[Q_i] < \mathbb{E}[Q_j]$ if $c_i > c_j$, and for $\nu > 1$, we have the opposite. More precisely, we have that $\mathbb{E}[Q_i] > \mathbb{E}[Q_j]$ if $\lambda_i > \lambda_j$, or if $c_i s_i > c_j s_j$. We can see as well that, as we mentioned above, in the non-saturated case, queue size does not depend on relative speed. It can be seen on the left plot

that the $\mathbb{C}_2$ and $\mathbb{C}_3$ have the same average, in spite of the fact that $\mathbb{C}_2$ is almost twice more powerfull than $\mathbb{C}_3$.

If, in the saturated case, a CE with c CPUs with relative speed s is equivalent to a CE with c/n CPUs with relative speed $s \cdot n$ (with $c/n \in \mathbb{N}$), this is not the case in non-saturated case: two CEs with the same number of CPUs are "equivalent" (in terms of queue size), whatever the relative speeds.

4 Used CPUs

The average number of used CPUs corresponds to the average number of jobs running on a Computing Element in $[0, t]$. In the sequential case, a simple argument allows to find this average, in the saturated case and in the non-saturate case. The first case is trivial: if the system is saturated and queues are almost never empty, each CPU is continuously processing a job. The average number of used CPUs on $\mathbb{C}_i$ is therefore c_i for any $\nu > 1$. For the second case, we know (see Lemma 3.1) that $\mathbb{E}[A_i(t)] \underset{t}{\sim} \lambda_i t$. Let's assume that at time t, the queue is empty, or at least with a size negligible regarding the $A_i(t)$. The (average) total amount of virtual work processed in $[0, t]$ tends therefore towards $\frac{\lambda_i t}{\mu}$ (the number of jobs multiplied by the average length), and so we need in average $\frac{\lambda_i}{\mu}$ virtual CPUs, or $\frac{\lambda_i}{\mu s_i}$ real (or effective) CPUs, which is equal to νc_i.

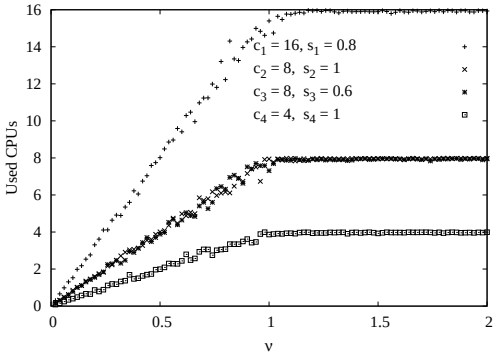

Figure 2: Average CPU usage.

As for queue size, we can easily obtain the total number of used CPUs (for $\nu > 1$): $\sum_i \nu c_i = \nu \sum_i c_i$.

The Figure 2 confirms the validity of our argument.

5 Resorption time

From a theoretical point of view, it can seem weird to consider a system where $\nu > 1$. Indeed, this kind of system is not stable, and its asymptotic behavior is not viable. We find however this analysis interesting, because it allows to answer the following question: if, starting from a point where all queues are (almost) empty, the load of our system is $\nu_1 > 1$ during a period t_1, and that after that period, the load goes down to $\nu_2 < 1$, how many time will it take before that all queues are (almost) completely resorbed? We are not giving below a precise proof, but rather a rough estimation of that resorption time.

We shown in Section 4, that if the load is $\nu < 1$, there will be in the average $c_i\nu$ (real) used CPUs, and then $c_i(1-\nu)$ free CPUs on $\mathbb{C}_i$. We know as well from Lemma 3.4 that if we have k CPUs, $\mathbb{C}_i$ will take in the average $\frac{N_i}{k\mu s_i}$ units of time for processing N_i jobs with average virtual length μ. We will then consider the following approximation giving the time needed by $\mathbb{C}_i$ for resorbing N_i sequential jobs with average virtual length μ', if the load is ν:

$$\frac{N_i}{c_i(1-\nu)\mu' s_i}$$

As a direct corollary of Theorem 3.2, we show that if there are N jobs waiting in the whole system, $\frac{Nc_is_i}{C}$ are in average on $\mathbb{C}_i$. The resorption does no longer depend on i, and we have:

Lemma 5.1 *The mean time needed for resorbing N sequential jobs (fairly distributed, with average length μ') if the load is $\nu < 1$ is approximately*

$$\frac{N}{C(1-\nu)\mu'}$$

In particular, this results means that if the input is suddenly stopped after having a load $\nu' > 1$ during t units of time ($N = \lambda t\frac{\nu'-1}{\nu'}$), the resorption time will be

$$\frac{\lambda t\frac{\nu'-1}{\nu'}}{C\mu'} = t(\nu'-1)$$

6 Slowdown

The slowdown for a particular job is classically defined as

$$\frac{\text{waiting time + execution time}}{\text{execution time}}$$

In our case, we will need to make a few adaptation to that formula, due to the heterogeneity of our system. We mainly want the same job with the same completion time (waiting time + effective execution time) on two CEs with different relative speeds having the same slowdown on those CEs. We assume that the user does not care that the real execution time of his job was shorter or longer that he though, he only cares about the time he waits before getting back his results.

We will then define the slowdown as (for a formal definition of effective and virtual execution time, cf. section 2.1, page 4):

$$\frac{\text{waiting time + effective execution time}}{\text{virtual execution time}}$$

If the average waiting time – that is, the time between the submission time and the begin of the job execution – for a job submitted on $\mathbb{C}_i$ at time θ with a load ν is $W_i(\theta,\nu)$, the average slowdown for a job submitted on $\mathbb{C}_i$ at time θ with a load ν will be

$$\mathbb{E}[\mathcal{SD}_i(\theta)] = \int_0^\infty \frac{W_i(\theta,\nu) + m/s_i}{m} f(m)dm = W_i(\theta,\nu)\int_0^\infty \frac{f(m)}{m}dm + {s_i}^{-1} \tag{3}$$

where $f(m)$ is the probability density for virtual job length. This average is valid only if waiting times are independent of the execution times, which is true for simple scheduling strategies such as FCFS or FF, but is not longer true for more complex techniques such as Backfilling [8].

If that function is not continuous, and that job lengths can take the values m_n, $n \in [1 \dots N]$, with a probability $P(m_n)$ the slowdown will be

$$\mathbb{E}[\mathcal{SD}_i(\theta)] = W_i(\theta,\nu)\sum_{n=1}^{N}\frac{P(m_n)}{m_n} + {s_i}^{-1}$$

6.1 Preliminary

Before splitting our argument in two cases ($\nu < 1$ and $\nu > 1$), we will compute the integral (or summation) in particular cases, keeping in mind first that $f(m)$ is a probability density function ($\int_0^\infty f(m)dm = 1$) and secondly that its average is μ ($\int_0^\infty mf(m)dm = \mu$).

We will use several distributions for the virtual execution time:
- Constant: each job has a constant virtual execution time of μ^{-1}.
- Uniform: virtual execution times have a uniform distribution going from $(1-\alpha)/\mu$ to $(1+\alpha)/\mu$ for α in $]0,1[$.
- Shifted exponential: as it is unfortunately not really possible to compute the average slowdown for an exponential distribution - the non-null probability density to have a zero length job makes the integral infinite - we will slightly modify the exponential distribution in the following way: the job length is a constant $\frac{\alpha}{\mu}$ plus an exponential distribution with a rate $\frac{\mu}{1-\alpha}$, with $\alpha \in]0,1[$.
- Erlang: virtual execution times have a one dimensional Erlang distribution with a shape h.

Table 6.1 summarizes the value of $\int_0^\infty \frac{f(m)}{m}$ for those distributions.

6.2 $\nu < 1$

Knowing the average waiting time (or the average queue size) is required in order to estimate the slowdown. Unfortunately, in the non-saturated case, we only have results for the Poissonnian case. Our results for the average slowdown are then also limited.

Table 1: summary of distributions used in this paper

Distribution	$f(m)$	$\int_0^\infty \frac{f(m)}{m}$
Constant	$N=1, m_1=\mu^{-1}, \mathbb{P}(m_1)=1$	μ
Uniform	$f(m)=\begin{cases}\frac{\mu}{2\alpha} & \text{if } \frac{1-\alpha}{\mu}<m<\frac{1+\alpha}{\mu}\\ 0 & \text{otherwise}\end{cases}$	$\frac{\mu}{2\alpha}\log\frac{1+\alpha}{1-\alpha}$
Shifted exponential	$f(m)=\begin{cases}0 & \text{if } m<\frac{\alpha}{\mu}\\ \frac{\mu}{1-\alpha}e^{-(m-\frac{\alpha}{\mu})\frac{\mu}{1-\alpha}} & \text{otherwise}\end{cases}$	$\frac{\mu}{1-\alpha}e^{\frac{\alpha}{1-\alpha}}\Gamma[0,\frac{\alpha}{1-\alpha}]$
Erlang	$f(m)=\frac{(\mu h)^h}{(h-1)!}m^{h-1}e^{-\mu m h}$	$\mu\frac{h}{h-1}$

Lemma 6.1 *If $\nu<1$, with shifted exponential distribution for job length, the average slowdown $\mathbb{E}[\mathcal{SD}_i]$ is asymptotically close to*

$$\frac{1}{c_i s_i}\ \frac{b(\nu c_i)^{c_i}}{c_i!(1-\nu)^2}\ \frac{e^{\frac{\alpha}{1-\alpha}}}{1-\alpha}\ \Gamma[0,\frac{\alpha}{1-\alpha}]+{s_i}^{-1}$$

Where b is the one defined in Theorem 3.1.

Proof : As in the section 3.1, for $\nu<1$, we first consider the Poisson-Exponential case.

We know by equation 3 that $\mathbb{E}[\mathcal{SD}_i(t)]=W_i(\theta,\nu)\int_0^\infty\frac{f(m)}{m}dm+{s_i}^{-1}$. In the poissonnian case, we can compute $W_i(\theta,\nu)$.

When there is at least one free CPU on $\mathbb{C}_i$, that is, when J_i is in $[0,c_i-1]$, the waiting time is null. If each CPU is processing some work and there is n (possibly 0) jobs in the queue, the arriving job will in the average wait $\frac{n+1}{\mu c_i}=\frac{\nu}{\lambda_i}(n+1)$ (because execution times are exponentially distributed). The average waiting time is then

$$\begin{aligned}W_i(\theta,\nu) &= \underbrace{\sum_{n=0}^{c_i-1}0\cdot\mathbb{P}[J_i=n]}_{=0}+\sum_{n=c_i}^{\infty}\frac{\nu}{\lambda_i}(n-c_i+1)\mathbb{P}[J_i=n]\\ &= \frac{\nu}{\lambda_i}\underbrace{\sum_{q=0}^{\infty}q\mathbb{P}[J_i=q+c]}_{=\mathbb{E}[Q_i]}+\frac{\nu}{\lambda_i}\underbrace{\sum_{q=0}^{\infty}\mathbb{P}[J_i=q+c_i]}_{=A}\end{aligned}$$

with (by equation 1)

$$\begin{aligned}A &= \sum_{q=0}^{\infty}\mathbb{P}[J_i=q+c_i]\ \underset{\theta}{\sim}\ \sum_{q=0}^{\infty}b\frac{\nu^{q+c_i}{c_i}^{c_i}}{c_i!}\\ &\underset{\theta}{\sim} b\frac{(\nu c_i)^{c_i}}{c_i!}\frac{1}{1-\nu}\end{aligned}$$

and (by Theorem 3.1)

$$\mathbb{E}[Q_i] = b\frac{\nu^{c_i+1}{c_i}^{c_i}}{c_i!(1-\nu)^2}$$

Then,

$$\begin{aligned} W_i(\nu,\theta) &\underset{\theta}{\sim} \frac{\nu}{\lambda_i}\left(b\frac{\nu^{c_i+1}{c_i}^{c_i}}{c_i!(1-\nu)^2} + b\frac{(\nu c_i)^{c_i}}{c_i!}\frac{1}{1-\nu}\right) \\ &\underset{\theta}{\sim} \frac{\nu}{\lambda_i}\frac{b(\nu c_i)^{c_i}}{c_i!(1-\nu)!} \end{aligned}$$

As quoted above, for a "pure" exponential distribution, the integral is infinite. However, our simulations have shown that if we slightly modify our job length distribution (we kept the inter-arrival distribution and we used a shifted exponential distribution for job lengths with a small α), queue sizes are still quite close to the $\mathbb{E}[Q_i]$ we obtained above. Despite that we cannot give an exact average or the asymptotic behavior of the slowdown, we are going to give an approximation of this average. We consider for that estimation that the $\mathbb{E}[Q_i]$ we got in Theorem 3.1 is still valid for a shifted exponential distribution for job lengths, if α is close enough to 0. We then give the approximation for this average:

$$\begin{aligned} \mathbb{E}[\mathcal{SD}_i] &\simeq \frac{\nu}{\lambda_i}\,\frac{b(\nu c_i)^{c_i}}{c_i!(1-\nu)^2}\,\frac{\mu}{1-\alpha}\,e^{\frac{\alpha}{1-\alpha}}\,\Gamma[0,\frac{\alpha}{1-\alpha}] + {s_i}^{-1} \\ &= \frac{1}{c_i s_i}\,\frac{b(\nu c_i)^{c_i}}{c_i!(1-\nu)^2}\,\frac{e^{\frac{\alpha}{1-\alpha}}}{1-\alpha}\,\Gamma[0,\frac{\alpha}{1-\alpha}] + {s_i}^{-1} \end{aligned}$$

Where b is the one defined in Theorem 3.1. □

We can see that, in case of real exponential distribution (that is, shifted distribution with parameter $\alpha = 0$), the average queue size is infinite, because $\Gamma[0,0]$ is infinite. However, the Γ function grows quite slowly: $\Gamma[0,\frac{\alpha}{1-\alpha}]$ is greater than 100 only for $\alpha < 10^{-43}$ ($\Gamma[0,\frac{\alpha}{1-\alpha}]$ is linear in $-\log(\alpha)$).

6.3 $\nu > 1$

In the system we are studying, we measure the slowdown of completed jobs. We computed then the average for each measured job[3]. But, at the end of our observation period, especially if $\nu \gg 1$, a lot of jobs are still in the queue and are therefore not taken into account in our average. The average slowdown is thus measured between the first job and the last finished job (and not the last submitted job) before the end of our observation period. Of course, the heavier the load, the sooner the last finished job has been submitted, and the fewer the number of jobs being part of the average.

[3]We are not fully assured that there is not a problem here: a job has to wait for another job completion, and thus slowdowns could be not independent. We still need to check that slowdowns are independent, and if not, if this could cause some problems.

In order to predict the average slowdown between the first job and the last one leaving the system before the end of the observation period, we will proceed in two steps; first we predict the average slowdown of the job submitted at a time θ ($\mathbb{E}[\mathcal{SD}_i(\theta)]$), then we will use these results for predicting the average slowdown on each job leaving the system between 0 and a time t, or the measured slowdown ($\mathbb{E}[\mathcal{MSD}_i(t)]$). Notice that, if $\nu < 1$, $\mathbb{E}[\mathcal{SD}_i(t)] \underset{t}{\sim} \mathbb{E}[\mathcal{MSD}_i(t)]$.

6.3.1 Slowdown for a job submitted at time θ.

Equation 3 gave us the general form of the slowdown for a job submitted at time θ. We compute in the previous section the value of the integral part for several job length distribution; we still need to estimate the waiting time $W_i(\theta, \nu)$.

Lemma 6.2 *if $\nu > 1$, the average waiting time for a job submitted at time θ is*

$$W_i(\nu, \theta) \underset{\theta}{\sim} \theta(\nu - 1)$$

Proof : From Theorem 3.2, we know that the average number of jobs in queue i is asymptotically (for $\theta \to \infty$)

$$\lambda_i \theta \frac{\nu - 1}{\nu}$$

The average time the N^{th} job in the queue i (c_i jobs are running, $N - 1$ jobs are waiting before this job) will wait tends (for $N \to \infty$) to $\frac{N}{\mu_i c_i}$. In order to be convinced of it, let us remark that the job flow at input of CPUs (that is, the flow at the output of queue of jobs coming in the CPUs) is in the average equal to the job flow at the output of CPUs. We therefore know that, in the average, the queue is emptied[4] according to $\mu_i t c_i$ (see Lemma 3.4), and thus that the N^{th} job waits in the queue in the average during $\frac{N}{\mu_i c_i}$ units of time.

Then, the average waiting time for a job coming at time t on $\mathbb{C}_i$ is (because $\frac{N}{\mu_i c_i}$ is linear in N)

$$W_i(\nu, \theta) \underset{t}{\sim} \lambda_i \theta \frac{\nu - 1}{\nu} \frac{1}{\mu_i c_i} = \theta(\nu - 1)$$

because $\frac{\lambda_i}{\mu_i c_i} = \nu$. □

Let remark that we found the same result as for the comment we did after Lemma 5.1, which is not surprising, because the resorption time after a time θ is equivalent to the waiting time to the last job submitted at that time.

And thus,

$$\mathbb{E}[\mathcal{SD}_i(\theta)] \underset{\theta}{\sim} \theta(\nu - 1) \int_0^\infty \frac{f(m)}{m} dm + {s_i}^{-1} \underset{\theta}{\sim} \theta(\nu - 1) \int_0^\infty \frac{f(m)}{m} dm \tag{4}$$

The following lemma is simply proved by replacing the integral in equation 4 by the results we got above (begin of section 6, page 13).

[4]That does not mean that the queue size goes down at that speed, but the jobs are leaving the queue at that speed.

Lemma 6.3 *if $\nu > 1$, the average slowdown of the job submitted at time θ is*

$$\mathbb{E}[\mathcal{SD}_i(\theta)] \underset{\theta}{\sim} \lambda\theta\frac{\nu-1}{\nu C} \cdot \begin{cases} 1 & (1) \\ \frac{1}{2\alpha}\log\left(\frac{1+\alpha}{1-\alpha}\right) & (2) \\ e^{\frac{\alpha}{1-\alpha}}\frac{\Gamma[0,\frac{\alpha}{1-\alpha}]}{1-\alpha} & (3) \\ \frac{h}{h-1} & (4) \end{cases}$$

if virtual execution times distribution is, respectively,
- (1) constant with value μ^{-1},
- (2) uniform going from $(1-\alpha)/\mu$ to $(1+\alpha)/\mu$ for α in $]0,1[$,
- (3) shifted exponential with parameter α,
- (4) Erlang with parameter h

6.3.2 Average slowdown until the last finished job with $\nu > 1$

We know the slowdown of a job submitted at a given time. We will now estimate the submission time of the job finishing at time θ, and compute the average slowdown between 0 and the submission time of the last finished job.

Lemma 6.4 *A job finishing at time τ has been submitted (in average) at time*

$$\theta = \frac{\lambda\tau - \nu C}{\lambda\nu}$$

Proof : If a job is submitted at a time θ in queue i, it will end up in the average at a time which is the sum of the arrival time, the average queuing time, and the average execution time: $\tau \underset{\theta}{\sim} \theta + \theta(\nu-1) + \mu^{-1} = \nu\theta + \frac{\nu C}{\lambda}$ □

In order to give an estimation of the average slowdown from 0 to $\frac{\lambda s_i t - \nu C}{\lambda s_i \nu}$ (that is, the average slowdown measured at time t), we need again to know the job length distribution as above.

Lemma 6.5 *If $\nu > 1$, the average slowdown for jobs leaving the system between 0 and t, in the case of constant execution time μ^{-1}, tends asymptotically on t towards*

$$\lambda t\frac{\nu-1}{2\nu^2 C}$$

Proof : We use here a reasoning close to the one we used for the estimation of the departure in queue size when $\nu > 1$. We "compact" our jobs in a way that we do not longer have idle periods in 0 and $t - \delta_k(t)$. That manipulation does not change the slowdown of any job, and thus keeps the average slowdown. Because the execution time is constant, we have on the k^{th} processor in the new system one job ending at time ${\mu_i}^{-1}$ (or $\frac{\nu C}{\lambda s_i}$), another one ending at time $2{\mu_i}^{-1}\ldots$ and one ending at time $\lfloor\mu_i(t-\delta_k(t)\rfloor{\mu_i}^{-1}$.

We know furthermore by Lemma 6.3 (1) and 6.4 that a job ending at time τ has been slowdowned in the average by

$$\frac{\lambda s_i \tau - \nu C}{\lambda s_i \nu} \frac{\lambda(\nu-1)}{C\nu} = \lambda \frac{\tau(\nu-1)}{\nu^2 C} + \frac{1-\nu}{s_i \nu} \underset{\tau}{\sim} \lambda\tau \frac{\nu-1}{\nu^2 C}$$

If $\mathcal{MSD}_i^k(t)$ is the measured slowdown on the k^{th} CPU of $\mathbb{C}_i$ between 0 and t, and $\mathcal{MSD}^{*k}_i(t')$ is the same in the modified system, we have then that (t' denotes $t - \delta_k(t)$)

$$\begin{aligned}
\mathbb{E}[\mathcal{MSD}_i^k(t)] &= \mathbb{E}[\mathcal{MSD}^{*k}_i(t')] \\
&\underset{t}{\sim} \frac{1}{\lfloor \frac{t'\lambda s_i}{\nu C} \rfloor} \sum_{j=1}^{\lfloor \frac{t'\lambda s_i}{\nu C} \rfloor} \left(j \frac{\nu C}{\lambda s_i} \frac{\lambda(\nu-1)}{\nu^2 C} \right) \\
&= \frac{1}{\lfloor \frac{t'\lambda s_i}{\nu C} \rfloor} \left(\frac{\nu-1}{s_i \nu} \frac{\lfloor \frac{t'\lambda s_i}{\nu C} \rfloor (\lfloor \frac{t'\lambda s_i}{\nu C} \rfloor + 1)}{2} \right) \\
&= \frac{\nu-1}{2\nu} (\lfloor \frac{t'\lambda}{\nu C} \rfloor + 1) \\
&\underset{t}{\sim} \frac{\nu-1}{2 s_i \nu} (\lfloor \frac{t\lambda s_i}{\nu C} \rfloor + 1) \qquad \text{because } t' \underset{t}{\sim} t \\
&\underset{t}{\sim} \lambda t \frac{\nu-1}{2\nu^2 C}
\end{aligned}$$

We can now compute the average measured slowdown for all our system $\mathbb{E}[\mathcal{MSD}_i(t)]$:

$$\begin{aligned}
\mathbb{E}[\mathcal{MSD}_i(t)] &= \frac{1}{c_i} \sum_{k=1}^{c_i} \mathbb{E}[\mathcal{MSD}_i^k(t)] \\
&\underset{t}{\sim} \frac{1}{c_i} \sum_{k=1}^{c_i} \lambda t \frac{\nu-1}{2\nu^2 C} \\
&= \lambda t \frac{\nu-1}{2\nu^2 C}
\end{aligned}$$

□

When the execution time is not constant, we will admit an approximation of the slowdown. We will consider that this time is really small compared to t and sufficiently regular, and replace the summation by an integral ($d\tau$ will be the execution time).

Lemma 6.6 *If $\nu > 1$, the average slowdown for jobs leaving the system between 0 and t, in the case of uniform execution distribution between $\mu(1-\alpha)$ and $\mu(1+\alpha)$ tends asymptotically on t towards*

$$\lambda t \frac{\nu-1}{2\nu^2 C} \cdot \frac{1}{2\alpha} \log\left(\frac{1+\alpha}{1-\alpha} \right)$$

Proof : In the case of uniform distribution, the job ending at time τ has been slowdowned by a time

$$\frac{\lambda\tau - \nu C}{\lambda\nu} \lambda \frac{\nu-1}{2\alpha C\nu} \log\left(\frac{1+\alpha}{1-\alpha} \right) = \log\left(\frac{1+\alpha}{1-\alpha} \right) \frac{\nu-1}{2\alpha\nu} \left(\frac{\lambda\tau}{C\nu} - 1 \right)$$

The average slowdown measured at time t can be approximated by (where t' is the instant of the last termination before, or just at, $t - \delta_k(t)$)

$$\begin{aligned} & \frac{1}{t'}\int_0^{t'} \log\left(\frac{1+\alpha}{1-\alpha}\right)\frac{\nu-1}{2\alpha\nu}\left(\frac{\lambda\tau}{C\nu}-1\right)d\tau \\ = \quad & \log\left(\frac{1+\alpha}{1-\alpha}\right)\frac{\nu-1}{2\alpha\nu}\left(\frac{\lambda t'}{2C\nu}-1\right) \\ \underset{t}{\sim} \quad & \lambda t \log\left(\frac{1+\alpha}{1-\alpha}\right)\frac{\nu-1}{4\alpha\nu^2 C} \end{aligned}$$

Because 1 is negligible compared to $\frac{\lambda t'}{2C\nu}$ and that $t' = t - \varepsilon - \delta_k(t) \underset{t}{\sim} t$, where ε is lower that the execution time of the first job ending after t, which is in the average μ, and does not depend upon t. □

Lemma 6.7 *If $\nu > 1$, the average slowdown for jobs leaving the system between 0 and t, in the case of shifted exponential distribution with parameter α tends asymptotically on t towards*

$$\lambda t \frac{\nu-1}{2\nu^2 C}\cdot e^{\frac{\alpha}{1-\alpha}}\frac{\Gamma[0,\frac{\alpha}{1-\alpha}]}{1-\alpha}$$

Proof : With the same argument, we found that in the case of a shifted exponential, the job ending at time τ has been slowdowned by a time

$$\frac{\lambda\tau-\nu C}{\lambda\nu}\lambda\frac{\nu-1}{\nu C}e^{\frac{\alpha}{1-\alpha}}\frac{\Gamma[0,\frac{\alpha}{1-\alpha}]}{1-\alpha}$$

and that the average slowdown measured at time t tends towards

$$\lambda t \frac{\nu-1}{2\nu^2 C}\cdot e^{\frac{\alpha}{1-\alpha}}\frac{\Gamma[0,\frac{\alpha}{1-\alpha}]}{1-\alpha}$$

□

Lemma 6.8 *If $\nu > 1$, the average slowdown for jobs leaving the system between 0 and t, in the case of Erlang distribution with shape h tends asymptotically on t towards*

$$\lambda t \frac{\nu-1}{2\nu^2 C}\cdot\frac{h}{h-1}$$

6.4 Experimental results

As for queue sizes, and for the same reasons, we show separately the cases $\nu < 1$ and $\nu > 1$. In the case $\nu < 1$, we only found results for (nearly) exponential distribution. Figure 3 (left) shows this case. For $\nu > 1$, we obtained results for different distributions (fixed, uniform, shifted exponential, and Erlang). Our predictions are superposed to our simulation results in those figures.

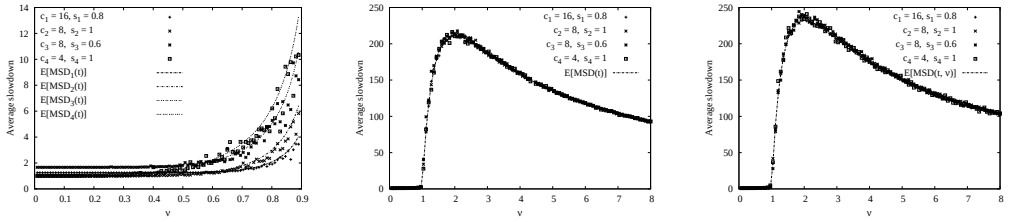

Figure 3: Average slowdown, with (left) exponential distribution shifted with $\alpha = 0.0005$ on non saturated system (Lemma 6.1), (center) fixed job length (Lemma 6.5)) on saturated system, and (right) uniform distribution from 0.5μ to 1.5μ on saturated system (Lemma 6.6)

The most interesting difference between $\nu < 1$ and $\nu > 1$ is probably that, in the first case, the average slowdown depends upon the CE on which the concerned job has been submitted on, while when $\nu > 1$, the average slowdown is the same for each CE. This appears in formula's: average slowdown for $\nu < 1$ contains i's, and average slowdown for $\nu > 1$ is not "i-dependent". However, we have to notice that we do not compute the average slowdown on the same number of jobs. For the first part, we compute the average on almost all submitted jobs, while in the second part, the bigger ν is, the more the proportion of measured job is small.

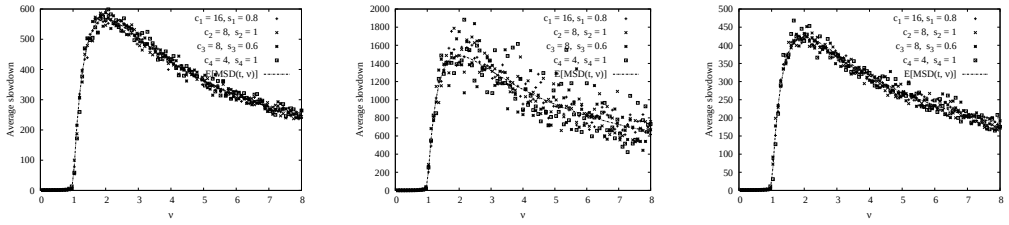

Figure 4: Average slowdown on saturated systems, with (left) shifted Exponential distribution with $\alpha = 0.05$, (center) idem with $\alpha = 0.0005$ (right) (Lemma 6.7), and (rigth) Erlang with shape (h)=2.

For $\nu < 1$, the slowdown we give corresponds to the average factor a job submitted at any time would be slowdowned. For $\nu > 1$, it is not the case; the slowdown for a job submitted at time t would be in average the one we gave in Lemma 6.3, but is still "i-independent"

As we could expect, when the job length distribution does not allow small jobs, the observed dispersion is rather small. In the case of shifted exponential distribution with a small α, (Figure 4, right plot), we observe some surprising extreme values; the values come generally from a really small job submitted late in the system. This kind of job waits a very long time in the queue (compared to its execution time), and has then a heavy weight in the average. In real situation, the same problem occurs when a job crashes at the very

beginning of its execution; that distorts the average slowdown. It is why Feitelson et al. [3] propose his *bounded slowdown*, which reduces the weight of small jobs.

Another remark is that the average slowdown (on finished jobs) is notably greater in some distributions than in others, despite the fact that, as we shown it above, queue sizes does not depend (too much) on job length distribution.

Figures 3 and 4 show how our predictions are close to our observations. We see that in case of shifted exponential, the smaller α, the bigger the dispersion.

7 Conclusions

Scheduling is one of the key services required for enabling performance on distributed and heterogeneous platforms such as computational grids. In this paper we have focused on a special kind of scheduler called a resource broker. A resource broker (also called a meta-scheduler) is mandatory when dealing with a multi-level architecture such as the one provided by the EDG/EGEE infrastructure. A resource broker dispatches requests on computing elements and each computing element schedules its own requests on its processors following a given policy (FIFO, etc.).

Due to the dynamic nature of many applications executed on computational grids, it is not possible to know in advance the arrival date and the duration of each request. Therefore, in this context it is neither possible to design a static algorithm that assumes the full knowledge of the application nor a dynamic one that only assumes the knowledge of task duration.

In this paper we have proposed and study a randomized resource broker for heterogeneous multi-level architectures where the arrival date and duration of the requests are given by probabilistic distributions (with fixed mean). We extend the results and analysis from previous literature (mainly from [1]) by generalizing the field of application studied so far. Our contribution is an extensive study of the behavior of the resource broker and the platform under such stochastic workload. More precisely we have studied the saturated and non-saturated case. For each case, we compute the average queue length of each computing elements and study the number of CPU used on each computing elements. We analyze the behavior of the system when it switches from the saturated to the non-saturated case. Finally, we evaluate how jobs are delayed according to the global load of the grid. Moreover, each of these metrics is studied both analytically and experimentally. To the best of our knowledge, those kind of analysis and measurment have only been studied in homogeneous environments; our work extends those results to heterogeneous systems.

We have shown by plotting together our simulation observations and our theoretical predictions or approximations that we have acquired a really good knowledge of job random brokering.

Our future works are directed towards more complex cases: current-state dependent brokering (based on queue lengths, free CPUs, estimation of waiting time,...), (partially) randomized or fully deterministic, for other job length and interarrival distributions. These new constraints will more than probably make our analysis more difficult. Indeed, for in-

stance, we will need to introduce a feedback from Computing Elements to the Resource Broker. We also want to extend our work towards fault tolerance and reliability by duplicating requests on several computing elements or by restarting request when a failure occurs.

References

[1] Vandy Berten and Joël Goossens. On the job distribution in random brokering for computational grids. In *Proceedings of Second International Symposium on Parallel and Distributed Processing and Applications (ISPA'2004)*. Springer-Verlag, 2004.

[2] C. Ernemann, V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yah-yapour. On Advantages of Grid Computing for Parallel Job Scheduling. In *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CC-GRID 2002)*, May 2002.

[3] Dror G. Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C. Sevcik, and Parkson Wong. Theory and Practice in Parallel Job Scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–34. Springer Verlag, 1997.

[4] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume II, 2d Ed. John Wiley & Sons, 1971.

[5] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *Proceedings of the First IEEE/ ACM International Workshop on Grid Computing*, pages 191–202. Springer-Verlag, 2000.

[6] M. Harchol-Balter. Task assignment with unknown duration. *Journal of the ACM*, 49(2):260–288, 2002.

[7] U. Sscwiegelshohn J. Krallmann and R. Yahyapour. On the design and evaluation of job scheduling algorithms. *Job Scheduling Strategies for Parallel Processing*, pages 17–42, 1999.

[8] A. Mu'alem and D. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Trans. Parallel and Distributed Syst.*, 12(6):529–543, June 2001.

[9] R Nelson. *Probability, Stochastic Processes, and Queueing Theory.* Springer-Verlag, 1995.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399