FIFO Queuing of Constant Length Fully Synchronous Jobs

Vandy Berten, Raymond Devillers and Guy Louchard

Département d'informatique, CP212
Université Libre de Bruxelles
B-1050 Bruxelles, Belgium
{vandy.berten,rdevil,louchard}@ulb.ac.be

Abstract. The paper examines the behaviour of a saturated multiprocessor system to which fully synchronous parallel jobs are submitted. In order to simplify the analysis, we assume constant length jobs, a non-preemptive scheduling and a FIFO queue. We determine in particular the number of used CPUs.
Keywords: FIFO scheduling, multiprocessor platforms, Next-Fit Bin-packing.

1 Introduction

In a (computational) Grid, clients submit their jobs to a job broker, who sends them to well chosen computing elements (or CE), using adequate deterministic or probabilistic strategies, based on static and/or dynamic information on the system and its history. To do so in a clever manner, it is necessary to have a good knowledge of the consequences of a choice, hence of the behaviour of a CE when jobs with some characteristics are sent to it at some rate. Since those behaviours are usually rather complex in realistic environments, it is common usage to make some (more or less justified) simplifications [5,6,1]. All of them simplify some job characteristics by models using standard stochastic processes or probabilistic distributions. Such characteristics are for instance arrivals or job lengths [5] or systems breakdowns [6].

Here, following typical grid features, we shall assume that a job may need more than one processor to perform its task, that successive jobs are independent from each others, that we know beforehand the distribution of the number of needed parallel processes (called the width of the job, assumed not to exceed the number s of processors of the CE), that the job is fully synchronous (i.e., all its processes have to start and finish simultaneously, because for instance they need to communicate in a synchronous fashion), that all the CPUs are identical, that due to the full synchrony of the jobs the CPU scheduling is non-preemptive, and that in order to avoid starvation problems the queue is managed as FCFS (or first come first served, or FIFO). Another popular strategy is FF (or first fit), which allow trespassings when the first jobs in the queue are too large for the current number of free CPUs, but smaller jobs are present further in the queue.

FF usually leads to a better CPU utilisation, but at the price that large jobs may be indefinitely delayed.

In order to further simplify the analysis, we shall assume that the system is saturated, i.e. there is always at least one job waiting in the input queue, and that all jobs have the same length, i.e. they last the same time, used as the time unit. It may seem curious to consider a saturated system, since then the queue will usually grow beyond any limit; the justification is manyfold; first, it is common that the load changes with time, and it may happen that during some periods the system is indeed saturated, but the non-saturated periods in between will allow to resorb the queue; next, our study will allow to determine when saturation does occur, and this is not trivial for multiprocessor jobs; finally, we plan to examine in more details the distribution of the number of used CPUs, and the saturated case provides an upper-bound approximation to the CPU usage. The unit length hypothesis, besides that it greatly simplifies the analysis, also tends to increase the CPU usage.

With those last hypotheses, we shall be allowed to divide the time in time slots of length 1. More exactly, initially (starting from an empty system, hence when the saturation hypothesis is not valid yet), we may have jobs starting at any time, but when the system will be saturated and if there is a non null probability to have jobs of width $\geq \lceil (n+1)/2 \rceil$, we are sure there will be a resynchronization, as illustrated on figure 1.

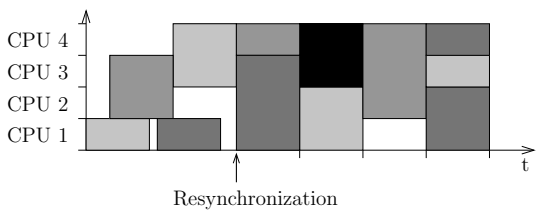

Fig. 1. Fixed length job, with resynchronization. After this resynchronization, the time line can be divided in slots.

To the best of our knowledge, this kind of system has not been studied in detail up to now. Of course, many other related problems remain to be further studied, like the behaviour of Bernoulli brokerings, index policies [1,7,4,3], ..., in order to define efficient brokering policies for a Grid with such servers. However, we do not aim here at finding an efficient strategy, but instead at studying the behaviour of saturated systems. Such a study could in a second step be helpful in finding efficient brokering strategies.

Moreover, the problem we consider is closely related to the problem of on-line algorithms for bin-packing, more exactly to the Next-Fit heuristics [2], which consists in putting items of various (monodimensional) sizes in bins, and in

opening a new bin as soon as the next considered item does not fit in the current bin. We compute here the average unused place in bins (and in some case the distribution) if the number of items tends towards the infinite.

The aim of this paper is then to computed the average number of used CPUs in a saturated system (receiving more work than what it is able to handle) composed of s CPUs, and where jobs are composed of several processes, each having a unitary execution length. In some configurations, we give the average number of used CPUs, and in some specific situations, we provide the full distribution, which means, at any time, the probability to have $k(\leq s)$ used CPUs.

This paper is structured as follows. In a first time, we consider two simple cases: the system is composed of 2 (section 2.1) and 3 (section 2.2) CPUs. Those two simple examples help in understanding the general case we present in section 2.3, where the system can have any size. In that general case, we provide the full distribution, but only in a non-closed form, i.e. in the form of a linear system which has to be solved for any chosen distribution.

We then present the distribution giving the worst CPU usage (section 2.4), for which we have a closed form for the distribution, and we finally present in section 2.5 the equidistributed case, where each size has an equal probability to occur.

2 Average number of used CPUs

Let us denote by w_n the probability that a job needs n CPUs ($n \leq s$) and by P_k^i the probability that at slot i, k CPUs are used. We define an n-job as being a job requiring n CPUs, and an n-slot as a slot where n CPUs are used.

In order to understand our argument, we will first consider two particular cases (when we have 2 and 3 CPUs), then we will analyze the general case.

2.1 Case $s = 2$

We will first have a look at the first slot. A 1-slot occurs at the first slot in only one case: the first job in the queue is a 1-job, immediately followed by a 2-job. For having a 2-slot, we have either one 2-job, or two 1-jobs. Then,

$$P_1^1 = w_1 w_2$$
$$P_2^1 = w_2 + w_1 w_1.$$

For other slots, the only possibility of having a 1-slot is that the previous slot was a 2-slot, and the two next jobs in the queue are respectively one 1-job (which runs during this slot) and one 2-job (which is the first waiting job during this slot).

In the case of a 2-slot, the previous slot is either a 1-slot, or a 2-slot. If the previous slot is a one slot, we know that the first waiting job is a 2-job. A 1-slot is then necessarily followed by a 2-slot. If the previous slot is a 2-slot, we have two possibilities: either the current job is a 1-job, or there are two 1-jobs running. We do not have any constraint about future jobs. Then,

$$\begin{cases} P_1^i = & P_2^{i-1} w_1 w_2 \\ P_2^i = P_1^{i-1} + & P_2^{i-1}(w_2 + w_1 w_1). \end{cases}$$

Or, in a matrix form:

$$\left(P_1^i \; P_2^i \right) = \left(P_1^{i-1} \; P_2^{i-1} \right) . \begin{pmatrix} 0 & 1 \\ w_1 w_2 & w_2 + {w_1}^2 \end{pmatrix}$$

Eigenvalues of this system are 1 and $w_1 w_2$, then this system converges towards the normalized ($P_1 + P_2 = 1$) solution of:

$$\left(P_1 \; P_2 \right) = \left(P_1 \; P_2 \right) . \begin{pmatrix} 0 & 1 \\ w_1 w_2 & w_2 + {w_1}^2 \end{pmatrix}$$

Or,

$$P_1 = \frac{w_1 w_2}{1 + w_1 w_2} \qquad \text{and} \qquad P_2 = \frac{1}{1 + w_1 w_2}.$$

The average number of used CPUs is then $1 \cdot P_1 + 2 \cdot P_2$, or $1 + \dfrac{1}{1 + w_1 w_2}$. In this case, the worst case of the number of used CPUs is reached when $w_1 = w_2 = \dfrac{1}{2}$, where $\dfrac{9}{5} = 1.8$ CPUs are used in the average.

2.2 Case $s = 3$

Again, we start with the first slot. The only case where only one CPU is used during the first slot is the one where the first job needs 1 CPU and the second one needs 3 CPUs, and is processed at the next slot. If the second job had needed 1 or 2 CPUs, it would have started at this first slot. If two slots are used at the first slot, we need either two monoprocessor jobs, or one bi-processor job, followed by a job that cannot fit in the remaining CPU. If all CPUs are used, there are four possibilities (three 1-job, one 1-job followed by one 2-job, one 2-job followed by one 1-job, or a 3-job), and we do not have any constraints on the first waiting job. We therefore have

$$\begin{aligned} P_1^1 &= w_1 w_3 \\ P_2^1 &= (w_1 w_1 + w_2)(w_2 + w_3) \\ P_3^1 &= w_1 w_1 w_1 + w_1 w_2 + w_2 w_1 + w_3 \end{aligned}$$

where w_k is the probability that a job asks for k CPUs.

Let us now look at other slots. It is possible to have one used CPU only if the previous slot was full, the first waiting job is a 1-job, and the second waiting job is a 3-job. If the previous slot weren't full, the first job would have been started at that previous slot, and if the next job is not a 3-job (i.e. either a 1- or a 2-job), it would have been started at the present slot.

The case "two used CPUs" is a bit more complex:

– The first waiting job during that slot is either a 2-job or a 3-job, because a 1-job would have been started at the current slot;
– It is not possible that the previous slot were a 1-slot, because we know that if a slot is a 1-slot, the first waiting job during that 1-slot execution is a 3-job;
– If the previous slot was a 2-slot, we already know that the first job waiting at the beginning of this slot is either a 2-job or a 3-job, out of which only the 2-case interests us (a 3-job would not lead to a 2-slot), with a conditional probability of $\frac{w_2}{w_2+w_3}$.
– If the previous slot is a 3-slot, we can either have a 2-job or two 1-jobs afterwards, and we have no previous knowledge.about the present situation.

The case 3-slot is rather similar:

– We do not have any constraint on the first waiting job during a 3-slot;
– The job waiting during a 1-slot is always a 3-job, so if the previous slot was a 1-slot, we already know that the current slot is a 3-slot;
– If the previous slot is a 2-slot, the first job in the queue is either a 3- or a 2-job, and the only ways to get 3 busy CPUs from these kinds of situation is either a 3-job, or a 2-job followed by a 1-job;
– If the previous slot is a 3-slot, we can have all combinations giving 3 used CPUs.

We can summarize for $i > 0$:

$$\begin{cases} P_1^i = & & P_3^{i-1} w_1 w_3 \\ P_2^i = & P_2^{i-1} \frac{w_2}{w_2+w_3}(w_2+w_3) & +P_3^{i-1}(w_2+w_1 w_1)(w_2+w_3) \\ P_3^i = P_1^{i-1} & +P_2^{i-1}\left(\frac{w_3}{w_2+w_3} + \frac{w_2 w_1}{w_2+w_3}\right) & +P_3^{i-1}({w_1}^3 + w_1 w_2 + w_2 w_1 + w_3). \end{cases}$$

In a matrix form, we have:

$$\begin{pmatrix} P_1^i & P_2^i & P_3^i \end{pmatrix} = \begin{pmatrix} P_1^{i-1} & P_2^{i-1} & P_3^{i-1} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & w_2 & \frac{w_3 + w_2 w_1}{w_2 + w_3} \\ w_1 w_3 & (w_2 + {w_1}^2)(w_2 + w_3) & {w_1}^3 + 2 w_1 w_2 + w_3 \end{pmatrix}$$

The matrix is clearly stochastic: every element is lower of equal to 1, and the sum of each row equals 1. Moreover, eigenvalues can easily be obtained with a computer tool such as *maple*. The largest one is 1, and the two others are in $]0,1[$. The system then converges towards the normalized ($P_1 + P_2 + P_3 = 1$) solution of:

$$\begin{pmatrix} P_1 & P_2 & P_3 \end{pmatrix} = \begin{pmatrix} P_1 & P_2 & P_3 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 1 \\ 0 & w_2 & \frac{w_3 + w_2 w_1}{w_2 + w_3} \\ w_1 w_3 & (w_2 + {w_1}^2)(w_2 + w_3) & {w_1}^3 + 2 w_1 w_2 + w_3 \end{pmatrix}$$

2.3 General Case

Let us first introduce some notations. Let β_k be the probability to have a succession of jobs using exactly k CPUs (with by definition $\beta_0 = 1$, corresponding to a empty sequence), and γ_k the probability for a given job that this job needs a number of CPUs greater or equal to k. We have, for $k \geq 1$,

$$\beta_k = \sum_{i=1}^{k} w_i \beta_{k-i}, \quad \text{and} \quad \gamma_k = \sum_{i=k}^{s} w_i.$$

We can now have a look at the first slot. The probability to be a k-slot ($k \leq s$) is the probability that the first jobs need k CPUs (β_k), and that the first waiting job during this slot cannot fit in the $s-k$ remaining CPUs (γ_{s-k+1}). Therefore,

$$P_k^1 = \beta_k \cdot \gamma_{s-k+1}.$$

And for a slot $i > 1$, we know that:

- The first job in the queue during this k-slot is either a $(s-k+1)$- or a $(s-k+2)$- ... or a s-job. This gives the γ_{s-k+1} of Equation 1;
- A k-slot can never follow a j-slot if $j \leq s-k$, otherwise the current slot jobs would have started at the previous slot. It is why the first summation in equation 1 goes from $j = s-k+1$ up to s;
- Let us assume that the previous slot is a j-slot with $s-k+1 \leq j \leq s$:
  - because the previous slot was a j-slot, we already know that the first job in the queue before the beginning of the i^{th} slot was either a $s-j+1$- or a $s-j+2$- ... or a s-job. That gives the γ_{s-j+1} below the fraction (conditional probability),
  - knowing that the first job in the queue before that slot needs between $s-j+1$ (see previous item) and k (the slot i is a k-slot) CPUs, we find directly the upper term of the fraction.

Therefore,

$$\begin{aligned} P_k^i &= \gamma_{s-k+1} \sum_{j=s-k+1}^{s} \left(P_j^{i-1} \frac{\sum_{\ell=s-j+1}^{k} w_\ell \beta_{k-\ell}}{\gamma_{s-j+1}} \right) \\ &= \gamma_{s-k+1} \sum_{j=s-k+1}^{s} \left(P_j^{i-1} \frac{\beta_k - \sum_{\ell=1}^{s-j} w_\ell \beta_{k-\ell}}{\gamma_{s-j+1}} \right). \end{aligned} \tag{1}$$

A deeper analysis should be done in the case where some w_i are null. Indeed, this could potentially lead to indeterminate values, if both numerators and denominators are null. However, we have several reasons for thinking that this will not cause problems:

– This could only lead to problem if the last values of w are null, or if $\exists K < s : w_i = 0\ \forall i \geq K$. If such a K exists, then $\gamma(k) = 0\ \forall k \geq K$. If some w_i are null but with non null probabilities for higher indices, this will not be problematic;
– We know that each factor multiplying P_j^{i-1} is a probability, then in $[0, 1]$, even if the factor is indeterminate;
– We observe experimentally that if some w_i are null for $i \geq K$ with the K defined here above, $P_k^1 = 0\ \forall k < S - K + 1$, and this 0 seems to propagate to further steps. Then each time a $\gamma = 0$ appears at the denominator, this is apparently as a factor of a $P_j^{i-1} = 0$, and can be ignored;
– It seems that if we do not consider $w_i = 0$ but $\lim_{w_i \to 0}$ instead, ambiguities can be resolved easily.

We let this more rigorous analysis to further research.

The initial condition may be expressed by saying that the slot before the first slot is always a s-slot; our initial condition is then:

$$P_k^0 = \begin{cases} 1 \text{ if } k = s, \\ 0 \text{ otherwise.} \end{cases}$$

and Equation (1) may be applied to the first slot as well (for $j = s$):

$$P_k^1 = 1\frac{\beta_k}{\gamma_1}\gamma_{s-k+1}.$$

and, because $\gamma_1 = 1$, we retrieve $P_k^1 = \beta_k \cdot \gamma_{s-k+1}$.

In a matrix form, Equation (1) can be rewritten as:

$$\left(P_1^i \cdots P_j^i \cdots P_s^i\right) = \left(P_1^{i-1} \cdots P_k^{i-1} \cdots P_s^{i-1}\right) \begin{pmatrix} 0 & 0 & \cdots & 0 & A_{1,s} \\ 0 & 0 & \cdots & A_{2,s-1} & A_{2,s} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & A_{s-1,2} & \cdots & A_{s-1,s-1} & A_{s-1,s} \\ A_{s,1} & A_{s,2} & \cdots & A_{s,s-1} & A_{s,s} \end{pmatrix} \quad (2)$$

where

$$A_{j,k} = \begin{cases} 0 & \text{if } j < s - k + 1, \\ \dfrac{\gamma_{s-k+1}}{\gamma_{s-j+1}} \left(\beta_k - \displaystyle\sum_{\ell=1}^{s-j} w_\ell \beta_{k-\ell}\right) & \text{otherwise.} \end{cases}$$

As A is stochastic, we know that 1 is an eigenvalue, and that every eigenvalue module is in $[0, 1]$. Hence, the convergence is ascertained if there are no other eigenvalues with 1 as module. We conjecture that this is true, as exhibited by numerous numerical checks.

Assuming that this system converges, it will converge towards the normalized ($\sum_{i=1}^{s} P_i = 1$) solution of the system $P = AP$. The average number of used CPUs becomes then $\sum_{i=1}^{s} iP_i$. Hence, we get

Theorem 1. *In the case of general job width distribution, with saturated system, if the job length is fixed and the system is synchronized, under the convergence hypothesis, the average number of used CPUs on a CE having s CPUs is*

$$\sum_{k=1}^{s} kP_k$$

where the P_k's are solutions of the linear system

$$\begin{cases} P_k = \displaystyle\sum_{j=s-k+1}^{s} \left(P_j \frac{\sum_{\ell=s-j+1}^{k} w_\ell \beta_{k-\ell}}{\gamma_{s-j+1}} \gamma_{s-k+1} \right) \\ \displaystyle\sum_{i=1}^{s} P_k = 1 \end{cases}$$

with $\beta_0 = 1$, $\beta_k = \displaystyle\sum_{i=1}^{k} w_i \beta_{k-i}$ and $\gamma_k = \displaystyle\sum_{i=k}^{s} w_i$.

2.4 Worst Case

As the average number depends upon the job width distribution, at least one distribution should give the worst average CPU usage. We do not have a formal proof about this worst distribution, but experiments we made let us believe that this distribution is pretty simple, and depends on the parity of the number s of CPUs.

In the case of odd s, we have $s = 2n + 1$ for some $n \in \mathbb{N}$. It seems in this case that the worst CPU usage is reached when $w_{n+1} = 1$ (and other ones are null). In this case, the average number of used CPUs is naturally $n + 1$. This is rather logically a "bad" distribution: if we had some narrower jobs, they could be run in parallel with a $(n + 1)$-job, and increase the number of used CPUs. If we had some wider jobs, when they would run (necessarily not in parallel with a $(n + 1)$-job), they use more CPUs than a $(n + 1)$-job, and then increase the average.

The case of even s ($s = 2n$) is a bit more complicated. It seems experimentally that the worst case is reached when two widths are possible: n and $n + 1$. We then have $w_n = x$ and $w_{n+1} = 1 - x$ for some x. Assuming this configuration is actually the worst, we can find the value of x minimizing the CPU usage.

With this distribution, we can only have n-, $(n + 1)$- and s-slots, the two firsts running resp. a n- and a $(n+1)$-job, the third one running two n-jobs. The only non null values of P are then P_n, P_{n+1} and P_s. With this distribution, we also get particular values for β and γ; a few computations yield that:

$$\beta_k = \begin{cases} 0 & \text{if } k < n \\ x & \text{if } k = n \\ 1 - x & \text{if } k = n + 1 \\ 0 & \text{if } n + 1 < k < s \\ x^2 & \text{if } k = s \end{cases} \quad \text{and} \quad \gamma_k = \begin{cases} 1 & \text{if } k \leq n \\ 1 - x & \text{if } k = n + 1 \\ 0 & \text{if } k > n + 1. \end{cases}$$

Now, non null values for the system $P = AP$ can be found:

$$P_n = P_n \cdot 0 + P_{n+1} \cdot x(1-x) + P_s \cdot x(1-x) \quad (3)$$
$$P_{n+1} = P_n \cdot 1 + P_{n+1} \cdot (1-x) + P_s \cdot (1-x) \quad (4)$$
$$P_s = P_n \cdot 0 + P_{n+1} \cdot x^2 + P_s \cdot x^2 \quad (5)$$

and the normalization equation

$$P_n + P_{n+1} + P_s = 1. \quad (6)$$

We have then:

$$\begin{aligned} P_n &= x(1-x)(P_{n+1} + P_s) && \text{from Eq. (3)} \\ &= x(1-x)(1-P_n) && \text{from Eq. (6)} \\ &= 1 - \frac{1}{1+x-x^2} \end{aligned}$$

$$\begin{aligned} P_{n+1} &= P_n + (P_{n+1} + P_s)(1-x) && \text{from Eq. (4)} \\ &= P_n + (1-P_n)(1-x) && \text{from Eq. (6)} \\ &= 1 - \frac{x}{1+x-x^2} \end{aligned}$$

$$\begin{aligned} P_s &= 1 - P_n - P_{n+1} && \text{from Eq. (6)} \\ &= 1 - \frac{1-x}{1+x-x^2}. \end{aligned}$$

The average number of used CPUs is

$$\begin{aligned} & nP_n + (n+1)P_{n+1} + sP_s \\ &= nP_n + (n+1)P_{n+1} + 2n(1 - P_n - P_{n+1}) \\ &= 2n - nP_n - (n-1)P_{n+1} \\ &= 2n - n(1-A) - (n-1)(1-xA) \quad \text{where } A = (1+x-x^2)^{-1} \\ &= 1 - A(x - nx - n). \end{aligned}$$

This average is minimal when $\frac{x - nx - n}{1 + x - x^2}$ reaches is maximum, which is the case when $x = \frac{\sqrt{n^2+n-1} - n}{n-1}$. If we inject this worst case x in the average, we get the following average:

$$\frac{2n(n+1) - 2 - n(n+1)\sqrt{n^2+n-1}}{2n(n+1) - 2 - (3n-1)\sqrt{n^2+n-1}}.$$

Notice that in the case of two CPUs ($n = 1$), the worst average is $\frac{9}{5}$, with $w_1 = w_2 = \frac{1}{2}$, as obtained before. And for large platforms (when n tends towards ∞), x tends towards 0; the worst case is then reached when there is only one possible width, just above the half of the number of servers $(n+1)$.

2.5 Equidistributed Case

While we do not have a general explicit solution for the system of Theorem 1 so far, we shall show that we can obtain an explicit result in the case of equidistributed lengths, that is, $w_i = w = \frac{1}{s}\ \forall i$. This equidistributed case is neither optimal nor a worst case in general, nor more realistic than other distributions, but simply it allows to go further. First of all, we have that, $\forall k > 1$

$$\begin{aligned}\beta_k &= w\sum_{i=1}^{k}\beta_{k-i}\\ &= w\sum_{i=0}^{k-1}\beta_i \\ &= w\beta_{k-1} + \underbrace{w\sum_{i=0}^{k-2}\beta_i}_{=\beta_{k-1}} = (w+1)\beta_{k-1}.\end{aligned} \tag{7}$$

Furthermore, we know that $\beta_0 = 1$ and $\beta_1 = w$, so, for $k \geq 1$

$$\beta_k = w(w+1)^{k-1}. \tag{8}$$

γ_k can be simplified as well: we find directly that $\gamma_k = (s-k+1)w$. We have now:

$$P_k^1 = \beta_k\gamma_{s-k+1} = w(w+1)^{k-1}wk = w^2(w+1)^{k-1}k,$$

and

$$P_k^i = \sum_{j=s-k+1}^{s}\left(P_j^{i-1}\frac{\sum_{\ell=s-j+1}^{k} w\beta_{k-\ell}}{jw}kw\right).$$

We have that

$$w\sum_{\ell=s-j+1}^{k}\beta_{k-\ell} = w\sum_{n=0}^{k-s+j}\beta_n \overset{\text{by (7)}}{=} \beta_{k-s+j+1} \overset{\text{by (8)}}{=} w(w+1)^{k-s+j-1}.$$

Therefore,

$$\begin{aligned}P_k^i &= \sum_{j=s-k+1}^{s}\left(P_j^{i-1}\frac{w(w+1)^{k-s+j-1}}{j}k\right)\\ &= \frac{w}{(w+1)^{s+1}}k(w+1)^k\sum_{j=s-k+1}^{s}\left(P_j^{i-1}\frac{(w+1)^j}{j}\right).\end{aligned}$$

We need now to show that this system converges. If we express that expression in a matrix form, we have the same form as Equation (2), with

$$A_{j,k} = \begin{cases} 0 & \text{if } j < s-k+1, \\ \frac{k}{j} w(w+1)^{k+j-s-1} & \text{otherwise.} \end{cases}$$

We can easily show that the matrix A is stochastic; we need for that, that each element of A is in $[0,1]$ and that the sum on a column is equal to 1. Because $A_{i,j}$ is obviously non-negative for each i, j, we just need to prove that $\sum_{k=1}^{s} A_{j,k} = 1, \forall j$.

$$\sum_{k=1}^{s} A_{j,k} = \frac{w}{j} \sum_{k=s-j+1}^{s} k(w+1)^{k+j-s-1} = \frac{w(w+1)^{j-s-1}}{j} \underbrace{\sum_{k=s-j+1}^{s} k(w+1)^{k}}_{=(1+w)^{s-j}(1+s)j}$$

$$= \frac{w}{j(1+w)}(1+s)j = \frac{\frac{1}{s}+1}{1+\frac{1}{s}} = 1.$$

The convergence of the system may be shown by an analysis similar to the one we shall conduct now, but as it is a bit lengthy and tedious, we do not include it here. We have

$$\begin{cases} P_k = k \frac{w}{(w+1)^{s-k+1}} \sum_{j=s-k+1}^{s} P_j \frac{(w+1)^j}{j} \ \forall k \in [1,s] \\ \sum_{k=1}^{s} P_k = 1. \end{cases}$$

The first equality is rewritten

$$\frac{P_k}{k}(w+1)^k = \sum_{j=s-k+1}^{s} \frac{w}{(w+1)^{s+1}} (w+1)^{2k} \frac{P_j (w+1)^j}{j}$$

We denote $\alpha_i = \frac{P_i}{i}(w+1)^i$ and $\psi = \frac{w}{(w+1)^{s+1}}$. Therefore,

$$\alpha_k = \psi (w+1)^{2k} \sum_{j=s-k+1}^{s} \alpha_j \tag{9}$$

and if $S = \sum_{j=1}^{s} \alpha_j$,

$$\alpha_k = \psi (w+1)^{2k} (S - \sum_{j=1}^{s-k} \alpha_j). \tag{10}$$

We want to express α_{s-i} instead of α_k:

$$\begin{aligned}
\alpha_{s-i} &= \psi(w+1)^{2(s-i)}\left(S-\sum_{j=1}^{i}\alpha_j\right)\\
&\overset{\text{by (9)}}{=} \psi(w+1)^{2(s-i)}\left(S-\sum_{j=1}^{i}\left(\psi(w+1)^{2j}\sum_{u=s-j+1}^{s}\alpha_u\right)\right)\\
&= \psi(w+1)^{2(s-i)}\left(S-\sum_{j=1}^{i}\left(\psi(w+1)^{2j}\sum_{u=0}^{j-1}\alpha_{s-u}\right)\right)
\end{aligned}$$

We denote $\tilde{\alpha}_i = \alpha_{s-i}$:

$$\begin{aligned}
\tilde{\alpha}_i &= \psi(w+1)^{2(s-i)}\left(S-\sum_{j=1}^{i}\left(\psi(w+1)^{2j}\sum_{u=0}^{j-1}\tilde{\alpha}_u\right)\right)\\
&= \psi(w+1)^{2(s-i)}\left(S-\psi\sum_{u=0}^{i-1}\sum_{j=u+1}^{i}(w+1)^{2j}\tilde{\alpha}_u\right).
\end{aligned}$$

We know that $\sum_{j=u+1}^{i}(w+1)^{2j} = \dfrac{(w+1)^2[(w+1)^{2i}-(w+1)^{2u}]}{w(w+2)}$. Therefore,

$$\tilde{\alpha}_i = \psi(w+1)^{2(s-i)}\left(S-\psi\sum_{u=0}^{i-1}\frac{(w+1)^2[(w+1)^{2i}-(w+1)^{2u}]}{w(w+2)}\tilde{\alpha}_u\right)$$

which is a linear recurrence with non constant coefficients. We then have to find the generating function $F(z)=\sum_{i=0}^{\infty} z^i\tilde{\alpha}_i$. Let us compute that function in several parts:

$$\begin{aligned}
\tilde{\alpha}_i &= \psi(w+1)^{2s}S\frac{1}{(w+1)^{2i}} - \psi^2\frac{(w+1)^{2s+2}}{w(w+2)}\sum_{u=0}^{i-1}\tilde{\alpha}_u\\
&\quad + \psi^2\frac{(w+1)^{2s+2}}{w(w+2)}\sum_{u=0}^{i-1}\frac{\tilde{\alpha}_u}{(w+1)^{2(i-u)}}
\end{aligned}$$

$\Rightarrow$

$$\sum_{i=0}^{\infty} z^i\tilde{\alpha}_i = \psi(w+1)^{2s}S\sum_{i=0}^{\infty}\left(\frac{z}{(w+1)^2}\right)^i \tag{11}$$

$$- \psi^2\frac{(w+1)^{2s+2}}{w(w+2)}\sum_{i=0}^{\infty}\sum_{u=0}^{i-1}(\tilde{\alpha}_u z^i) \tag{12}$$

$$+ \psi^2\frac{(w+1)^{2s+2}}{w(w+2)}\sum_{i=0}^{\infty}\sum_{u=0}^{i-1}\frac{z^i\tilde{\alpha}_u}{(w+1)^{2(i-u)}} \tag{13}$$

With a few computations,

$$(11) = \psi(w+1)^{2s} S \sum_{i=0}^{\infty} \left(\frac{z}{(w+1)^2} \right)^i = \psi(w+1)^{2(s+1)} S \frac{1}{(w+1)^2 - z}$$

$$(12) = \psi^2 \frac{(w+1)^{2s+2}}{w(w+2)} \sum_{i=0}^{\infty} \sum_{u=0}^{i-1} (\tilde{\alpha}_u z^i) = \psi^2 \frac{(w+1)^{2s+2}}{w(w+2)} \frac{z}{1-z} F(z)$$

$$(13) = \psi^2 \frac{(w+1)^{2s+2}}{w(w+2)} \sum_{i=0}^{\infty} \sum_{u=0}^{i-1} \frac{z^i \tilde{\alpha}_u}{(w+1)^{2(i-u)}} = \psi^2 \frac{(w+1)^{2s+2}}{w(w+2)} \frac{z}{(w+1)^2 - z} F(z)$$

Therefore,

$$F(z) = \psi(w+1)^{2(s+1)} S \frac{1}{(w+1)^2 - z} - \psi^2 \frac{(w+1)^{2(s+1)}}{w(w+2)} \frac{z}{1-z} F(z)$$
$$+ \psi^2 \frac{(w+1)^{2(s+1)}}{w(w+2)} \frac{z}{(w+1)^2 - z} F(z)$$

From which we can obtain

$$F(z) = Sw(w+1)^{1+s} \frac{1-z}{(1+w-z)^2}$$

Finally, we have

$$\tilde{\alpha}_n = w(w+1)^{s-1} S \frac{1-wn}{(w+1)^n}$$

We can now extract P_k from $\tilde{\alpha}_k$'s definition:

$$P_k = \frac{\tilde{\alpha}_{s-k} k}{(w+1)^k} = \frac{w(w+1)^{s-1} S \frac{1-w(s-k)}{(w+1)^{s-k}} k}{(w+1)^k} = (wk)^2 (w+1)^{-1} S = \frac{Sw^2}{w+1} k^2$$

We know furthermore that $\sum_{k=1}^{s} P_k = 1$, and that $w = \frac{1}{s}$. We can then find S:

$$1 = \sum_{k=1}^{s} \frac{Sw^2}{w+1} k^2 = \frac{Sw^2}{w+1} \frac{2s^3 + 3s^2 + s}{6} = \frac{S}{1+s} \frac{2s^2 + 3s + 1}{6}$$

Therefore,

$$S = \frac{6(1+s)}{2s^2 + 3s + 1} = \frac{6}{2s+1}$$

and

$$P_k = \frac{6k^2}{s(1+2s)(1+s)}$$

and

$$\mathbb{E}_k[P_k] = \frac{3s(1+s)}{2(1+2s)}$$

Finally, we obtain that the average number of free CPUs is

$$s - \mathbb{E}_k[P_k] = \frac{s(s-1)}{2(1+2s)}$$

which is compatible with the numerical resolutions we made on A, and the simulations we made. The proportion of free CPUs is then

$$\frac{s-1}{2(1+2s)}$$

which tends towards $\frac{1}{4}$ when s tends towards ∞. We then get

Theorem 2. *In the case of equidistributed job width distribution between 1 and s (the CE size), with saturated systems, if the job length is fixed, the average number of used CPUs is*

$$\frac{3s(s+1)}{2(1+2s)}.$$

This last result allows to determine the saturation point $\tilde{\nu}$ of such a system, i.e., the normalized arrival rate from which the system becomes (possibly after some transitory phase) saturated. If the average job arrival rate is λ, $\mathcal{W} = \sum_k k w_k$ is the average job width and M is the average job length (here $M = 1$, from the hypotheses and the choice for the time unit), the normalized arrival rate is defined as $\nu \triangleq \frac{\lambda.\mathcal{W}.M}{s}$. It is well-known that, for sequential jobs (i.e., when $\mathcal{W} = 1$), the saturation point is given by $\tilde{\nu} = 1$, but when parallel jobs are allowed, it occurs (generally slightly) before 1. We then get

Theorem 3. *In case of equidistributed job width distribution between 1 and s (the CE size), if the job length is fixed, the point of saturation $\tilde{\nu}$ is*

$$\frac{3(s+1)}{2(1+2s)}.$$

Proof. It is known that, for an unsaturated system, i.e. when $\nu \leq \tilde{\nu}$, the average number of used CPUs is νs, and from theorem 2, it is $\frac{3s(s+1)}{2(1+2s)}$ for a saturated system, i.e. when $\nu > \tilde{\nu}$; $\tilde{\nu}$ can now easily be found (in the case of constant execution time, and equidistributed job width between 1 and s) as being the intersection between those to lines:

$$\tilde{\nu} s = \frac{3s(s+1)}{2(1+2s)}.$$

Therefore,

$$\tilde{\nu} = \frac{3(s+1)}{2(1+2s)}.$$

■

Experiments we made have confirmed these theoretical predictions. Notice that this saturation point equals of course 1 when $s = 1$, and tends towards $\frac{3}{4}$ when s tends towards ∞.

3 Conclusion

We have analyzed the behaviour of a FIFO queue for a saturated multiprocessor system with fully-synchronized fixed length parallel jobs. We have characterized the distribution of the number of used CPUs, solved it completely for bi- or tri-processors, determined the worst case and produced a closed formula for the distribution (hence for the average) of the number of used CPUs in the case of equidistributed job width, determining as a consequence the saturation point in this last case.

The main contribution of this paper is a deep analysis of the maximal utilization of a platform composed of several processors, on which fully-synchronous parallel jobs run. Such an analysis is very useful for dimensioning a computing system, or in order to tune the (meta-)scheduler.

The studied model is rather simple, and more realistic models should be considered in a near future. For instance by raising the execution length hypothesis, and by considering other models, such as for instance exponential, Log-Normal or Log-Uniform service times. However, these extensions will certainly require a rather different approach.

References

1. Vandy Berten. *Stochastic Approach to Brokering Heuristics for Computational Grids*. PhD thesis, Université Libre de Bruxelles, 2007.
2. E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson *Approximation Algorithms for Bin-Packing – An Updated Survey* . In Algorithm Design for Computer System Design, ed. by Ausiello, Lucertini, and Serafini. Springer-Verlag, 1984.
3. Ernemann, C., Hamscher, V., Schwiegelshohn, U., Streit, A., and Yahyapour, R. *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).
4. Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. *Theory and Practice in Parallel Job Scheduling*. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph, Eds. Springer Verlag, 1997, pp. 1–34.
5. Emmanuel Medernach, *Workload Analysis of a Cluster in a Grid Environment*, in Job Scheduling Strategies for Parallel Processing, Lect. Notes Comput. Sci. vol. 3834, pp 36–61, 2005
6. N. Thomas, J. Bradley and W. Knottenbelt, *Stochastic Analysis of Scheduling Strategies in a Grid-based Resource Model*, IEE Proceedings-Software 151:5, pp 232–239, 2004.
7. Richard R. Weber and Gideon. Weiss, *On an index policy for restless bandits*, Journal of Applied Probability, vol 27, pp 637-648, 1990.