



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

*Performance Analysis of Work Stealing for
Streaming Systems and Optimizations*

Jonatha Anselmi — Bruno Gaujal

N° 6988

Juillet 2009

*Rapport
de recherche*

Performance Analysis of Work Stealing for Streaming Systems and Optimizations

Jonatha Anselmi* , Bruno Gaujal

Thème : Calcul distribué et applications à très haute performance
Équipes-Projets Mescal

Rapport de recherche n° 6988 — Juillet 2009 — 17 pages

Abstract: This paper studies the performance of parallel stream computations on a multiprocessor architecture using a work-stealing strategy. Incoming tasks are split in a number of jobs allocated to the processors and whenever a processor becomes idle, it steals a fraction (typically half) of the jobs from a busy processor. We propose a new model for the performance analysis of such parallel stream computations. This model takes into account both the algorithmic behavior of work-stealing as well as the hardware constraints of the architecture (synchronizations and bus contentions). Then, we show that this model can be solved using a recursive formula. We further show that this recursive analytical approach is more efficient than the classic global balance technique. However, our method remains computationally impractical when tasks split in many jobs or when many processors are considered. Therefore, bounds are proposed to efficiently solve very large models in an approximate manner. Experimental results show that these bounds are tight and robust so that they immediately find applications in optimization studies. An example is provided for the optimization of energy consumption with performance constraints. In addition, our framework is flexible and we show how it adapts to deal with several stealing strategies.

Key-words: Work Stealing, Performance Evaluation, Markov Model

* sceptre

Analyse des Performances du vol de travail pour des systèmes de flux et applications à la consommation énergétique

Résumé : Dans cet article nous étudions les performances des calculs dans un système de flux qui s'exécute sur une architecture multi-processeurs utilisant une stratégie de vol de travail: les tâches sont découpées en des sous-tâches allouées aux processeurs, et dès qu'un processeur devient inactif, il vole une fraction du travail restant à un processeur encore actif. Nous proposons un nouveau modèle pour étudier les performances qu'un tel système. Ce modèle prend en compte à la fois les aspects logiciels et matériels (conflits sur le bus de communications, synchronisations) du système. Ensuite, nous montrons une approche analytique récursive qui est plus efficace que la technique classique de la résolution des équations d'équilibre. Cependant, cette approche reste numérique inapplicable quand les tâches se découpent en un grand nombre de sous-tâches et quand le nombre de processeurs est très grand. Dans ce cas, nous proposons des bornes rapides sur le temps d'attente des tâches qui sont de bonne qualité et qui peuvent être utilisées dans des problèmes d'optimisation (optimisation paramétrique du vol de travail et consommation énergétique).

Mots-clés : Vol de Travail, Evaluation de Performances, Modèle Markovien

1 Introduction

Modern embedded systems perform on-the-fly real-time applications, (e.g., compress, cipher or filter video streams) whose computational complexity requires using multiprocessor architectures (in terms of FLOPS as well as energy consumption). This paper is concerned with such systems where stream computations are processed by a multiprocessor architecture using a work-stealing scheduling algorithm. We take our inspiration from an experimental board developed by ST Microelectronics (Traviata) over the STM8010 chip. The chip is composed of three almost identical ST231 processors communicating via a multicom network. This board is used as an experimental platform for portable video processing devices of the near future [1]. What we call a stream computation here can be modeled as a sequence of independent *tasks* characterized by their arrival times and their sizes that may vary, e.g., a video stream under Mpeg coding. As for the system architecture, it is modeled by a multiprocessor system interconnected by a very fast communication network, typically a fast bus. The system functions according to the following work-stealing principle that is specific to streaming applications :

- Each incoming *task* is split into atomic independent *jobs* that are evenly distributed among the processors (think of an image split into pixels for processing).
- As soon as a processor becomes idle, it steals a fraction of the remaining jobs from the processor with the current largest backlog of jobs.
- Once all jobs of one task have been executed, the task is completed and the same process can start over with a new task, provided that there is one available in the system buffer.

Generally speaking, work stealing is a scheduling policy where idle resources steal jobs from busy resources; see [16, 4, 8, 3] for an exhaustive overview of related work. The work stealing paradigm has been implemented in several parallel programming environment such as Cilk [10] and Kaapi [13, 2]. The success of the work-stealing paradigm is due to the fact that it has many interesting features. First, this scheduling policy is very easy to implement and does not require many information on the system to work efficiently, because it is only based on the current state of each processor (idle or not). Second, it is asymptotically optimal in terms of worst-case complexity [5]. Finally, it is *processor oblivious* since it automatically adapts on-line to the number and the size of jobs in the system as well as to the changing speeds of processors [7].

Many variants of work stealing have been developed. In the following, we will consider a special case of the work-stealing principle introduced above: at each steal, half of the remaining work is stolen from the busiest processor. Let n_r be the number of unit jobs initially assigned to processor $1 \leq r \leq R$, with speed μ_r . It should be clear that after R steals the maximum backlog is cut by at least half so that the total number of steals is upper bounded by $R \log_2(\max_r n_r)$ and if γ is the time needed for one steal, then by summing, the completion time C satisfies: $\frac{\sum_r n_r}{\sum_r \mu_r} \leq C \leq \frac{\sum_r n_r + \gamma R \log(\max_r n_r)}{\sum_r \mu_r}$. In [4] similar bounds holding with high probability are provided for a more general case where the victim is chosen at random. However, to the best of our knowledge, very few performance evaluations of work-stealing have been proposed in the literature that take into account higher moments of the completion time that are needed to estimate mean performance measures such as waiting time. Also, few studies use accurate stochastic models taking into account hardware constraints such as exclusive communications among processors (bus contentions) as well as software features (stealing jobs from the busiest processor). In [6] the generic work-stealing principle is proven to be stable when the input rate is smaller than the processing rate but no quantitative formulas of its performance are given. Scheduling policies where idle processors steal from busy processors are analyzed in [17]. However, the latter approach does not take into account the synchronizations and assumes that all processors are independent, unlike what is done here.

In this paper, we propose a two-level model for a streaming system evolving in a changing environment. At the task level, the system is reduced to a simple queueing system (M/G/1 queue) so that the Pollaczek–Khintchine formula, e.g., [9], can be used to assess the mean performance of the system provided that the mean and the second moment of the (task) service time distribution can be computed. At the job level, the system is modeled as a continuous time Markov chain whose transitions correspond to job services or steals. This is used to compute the first two moments of the service time distribution useful for the task level model. We show that this approach drastically reduces the computational requirements of the classic global balance technique, e.g., [9]. However, it remains computationally impractical when tasks split in many jobs and when many processors are considered. Therefore, we propose efficient bounds aimed at quickly obtaining the model solution in an approximate manner. With respect to mean waiting times, experimental results show that the proposed bounds are very tight and robust capturing very well the dynamics of the work-stealing paradigm above. The analytical simplicity of the proposed

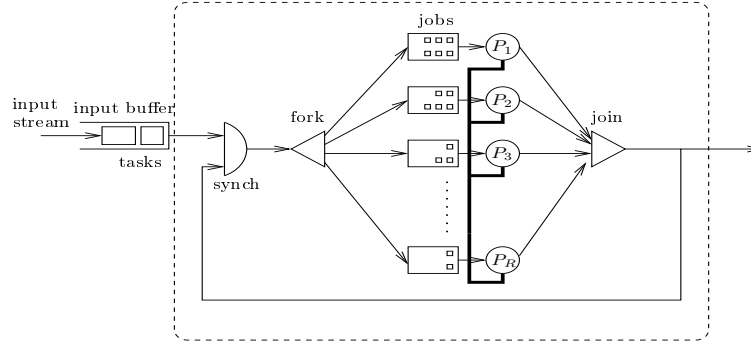


Figure 1: The fork-join nodes are coupled: the join has to wait for all jobs to be treated before sending a signal back and releasing one pending task. The synch node ensures that tasks are treated in sequence, i.e., no processor starts executing a job of a given task if there exists a job of its previous task which is still handled by some processor. The thick lines represent the interconnection network that is used to steal in a mutual exclusive manner. The global system can be seen as one M/G/1 queue when the dotted area is seen as one processor for tasks.

bounds lets us devise a convex optimization model determining the optimal number of processors and processing speeds which minimize energy consumption while satisfying a performance constraint on the mean waiting time. We also show how our framework adapts to different stealing strategies aimed at balancing the load among processors. The goodness of these strategies turns out to strongly depend on the structure of communication costs so that their impact is non-trivial to predict without our model.

The paper is organized as follows. In Section 2 we describe the real-world problem and develop a queueing model taking into account work-stealing effects as well as hardware constraints. In Section 3 we propose an analytical framework providing the exact solution of our model and compare its computational complexity with the one of standard global balance. Section 4 proposes efficient bounds on the performance indices of interest, and Section 5 illustrates numerical results validating the accuracy of our approach. Section 6 presents an optimization model for determining the optimal number of processors and speeds which minimize energy consumption costs, and Section 7 discusses how our framework can be applied to deal with different stealing strategies. Finally, Section 8 draws the conclusions of our work and outlines future research.

2 Model of Work Stealing over a Multi-processor Architecture

To assess the performance of the systems introduced above, one must take into account both the algorithmic features of work-stealing and the hardware constraints of the architecture. The system presented in Figure 1 fits rather well the Traviata multiprocessor dsp system developed by ST Microelectronics for streaming video codec [1] where tasks are images and jobs are local processing algorithms to be performed on each pixel (or group of pixels) of the image.

The main difficulty here is the fact that the system contains several types of synchronizations: the rendez-vous point among incoming tasks and released tasks (the *synch* node in Figure 1) as well as the *fork* and *join* nodes. Also, at the hardware level, bus contention imposes mutual exclusion for steals and the global synchronization among processors is state dependent. Indeed, the steal can only happen when one buffer is empty, the number of stolen jobs also depends on the current state, and only the *busiest* processor is stolen. All these latter features are hard to model using compact models but can be taken into account at the state-space level in a Markov chain. This is why our model is constructed on two levels. At the task level, our model can be represented by a queueing network with fork and join nodes, as done in Figure 1. At the job level, a direct Markov chain model is used.

Here, we assume that all timings are random with exponential distributions. The hardware constraints impose that two steals cannot happen at the same time because of contentions on the communication bus among the processors and that the processing over one task can only start when the previous task is completely finished (no pipelining is allowed).

At the task level, the system behavior is illustrated by the queueing model presented in Figure 1. The fork and join nodes ensure that jobs are spread over all processors. Note that the work-stealing behavior of the processors is not represented in the figure. As for the detailed behavior of the processors, this is modeled by a reducible continuous time Markov chain that is described in the next section.

According to the description above, task i can be executed if and only if each job of task $i - 1$ have finished their execution.

2.1 Queuing Model

We now introduce a queueing model for the system displayed in Figure 1, to capture the performance dynamics of the real-world problem introduced above. It is composed of R service units (or processors) and each service unit r has a local buffer. If not otherwise specified, indices r and s will implicitly range in set $\{1, \dots, R\}$ indexing the R processors. We assume that *tasks* arrive from an external source according to a Poisson process with rate λ . When a task enters the system, it splits into $N_k \cdot R$ independent *jobs*, $N_k \in \mathbb{Z}^+$, with probability p_k , $k = 1, \dots, K$, and, for simplicity, these jobs are equally distributed among all processors, that is N_k jobs per processor (any initial unbalanced allocation can also be taken into account with minimal changes in the following results). This models the fact that tasks can have different sizes or jobs can be encoded in different ways. Processors are assumed to handle jobs according to a First-Come-First-Served (FCFS) service discipline. The service time required by the execution of a job assigned to processor r is a random variable exponentially distributed with mean μ_r^{-1} . If task i joins the system when a processor is still executing jobs from task $i - 1$, then it is stored in a queue of *blocked* tasks, i.e., the input buffer of Figure 1. When all jobs of task $i - 1$ have been processed, all jobs of task i (is present in the input buffer) are equally distributed among all processors in turn. During the execution of task i , if processor r becomes idle, then it attempts to steal $\lfloor n_{\max}/2 \rfloor$ jobs from the queue of the processor with the largest number of jobs, i.e., n_{\max} . Such stealing actions cannot be performed if no processor has backlogged jobs, because jobs are atomic units. When a processor steals jobs from the queue of another processor, it uses the communication bus in an exclusive manner (no concurrent steal can take place simultaneously). This further incurs a *communication cost* which depends on the number of jobs to transfer (more details of this point are provided in Section 7). This is interpreted as the time required to transfer jobs between the processor queues. We assume that the time needed by a processor to probe the local queues of the other processors is negligible. This is because multiprocessor embedded systems are usually composed of a limited number of processors. We also assume that communication costs are exponentially distributed random variables with mean γ_i^{-1} where $\lfloor i/2 \rfloor$ represents the number of jobs to transfer.

Let $\mathbf{n}(t) = (n_1(t), \dots, n_R(t))$ be the vector denoting the number of jobs in each internal buffer at time t . In order to model bus contention, we consider the case where only one processor at a time is allowed to perform a steal if several processors are idle. We assume the following rule stating which processor is allowed to steal. Any other rule to select one idle processor will have a negligible impact on the performance.

Assumption 1 *In $\mathbf{n}(t)$, if more than one processor can steal jobs from the queue of processor r , i.e., $\|\{s : n_s = 0\}\| > 1$, then only processor $\min\{s : n_s = 0 \wedge s > r\}$ is allowed to perform the steal from r if it exists. Otherwise the jobs are stolen by $\min\{s : n_s = 0 \wedge s < r\}$.*

On the other hand, when processor r can steal jobs from more than one processor, we also make the following assumption stating which processor is stolen.

Assumption 2 *In $\mathbf{n}(t)$, if $\|\{s : n_s = \max_r n_r\}\| > 1$, then jobs can be stolen only from the queue of processor $\min\{s : n_s = \max_r n_r\}$.*

Under the foregoing assumptions and assuming, for simplicity, $K = 1$, the state of the system

$$\{(m(t), \mathbf{n}(t)) : m(t) \geq 0, \mathbf{n}(t) \geq \mathbf{0}\}_{t \in \mathbb{R}^+} \quad (1)$$

is a stochastic process describing the number of tasks waiting for service at time t in the system (i.e., m), and the number of jobs associated to each processor, (i.e., \mathbf{n}), in the proposed model. It is direct to see that (1) is a continuous-time Markov chain. The goal of our study is to provide efficient analysis for (1) to compute the value of stationary performance indices, i.e., when $t \rightarrow \infty$, such as the mean task waiting time and the mean number of tasks in the system. For simplicity, in the following we omit the word "stationary" when we refer to performance indices.

Let $\mathbf{n} \in \mathbb{Z}^R$. For simplicity, we summarize the notation used in the remainder of the paper:

R	:=	number of processors,
λ	:=	mean task arrival rate,
p_k	:=	probability that incoming tasks split in $N_k R$ jobs, $N_k \in \mathbb{Z}^+$, $k = 1, \dots, K$,
μ_r^{-1}	:=	mean service time of processor- r jobs,
μ	:=	$\sum_{r=1}^R \mu_r$,
γ_i^{-1}	:=	mean communication cost (delay) incurred transferring $\lfloor i/2 \rfloor$ jobs,
\mathbf{n}	:=	(n_1, \dots, n_R) , vector denoting the number of jobs in each processor,
\mathbf{e}_r	:=	unit vector in direction r ,
X_r	:=	exponential random variable with mean μ_r^{-1} ,
X	:=	exponential random variable with mean μ^{-1} ,
$\mathcal{T}_{\mathbf{n}}$:=	task service time when n_r jobs are assigned to processor r , $\forall r$,
$T_{\mathbf{n}}$:=	$\mathbb{E}[\mathcal{T}_{\mathbf{n}}]$,
$V_{\mathbf{n}}$:=	$\mathbb{E}[\mathcal{T}_{\mathbf{n}}^2]$,
\mathcal{T}	:=	task service time,
T	:=	mean task service time, i.e., $\mathbb{E}[\mathcal{T}]$,
V	:=	second moment of task service time, i.e., $\mathbb{E}[\mathcal{T}^2]$,
W	:=	mean task waiting time.

3 Performance Analysis Framework

We now develop an exact analysis for the performance analysis of the work-stealing model introduced in the previous section. We initially observe that the exact solution of the proposed model can be obtained by applying classic global balance equations, e.g., [9], of the underlying Markov chain (1). However, this requires a truncation of the Markov chain state space and the solution of a prohibitively large linear system. This motivates us to investigate alternative approaches for obtaining the exact model solution.

The key point of our approach consists in computing the first two moments of the service time distribution of each task in order to define a M/G/1 queue and obtain performance indices estimates at the task level by exploiting standard formulas. This is possible because we remark that jobs belonging to different tasks do not overlap with each other, i.e., they are independent, because jobs belonging to task i can be processed if and only if *all* jobs of task $i - 1$ have finished their execution. This approach provides an alternative analytical framework able to characterize the exact solution of the proposed work-stealing model without applying standard (computationally impractical) techniques for solving continuous-time Markov chains.

3.1 Exact Analysis

In our framework, we remark that This suggests the alternative approach of modeling the (exact) *global* service time distribution of each task, i.e., the time needed by the multiprocessor system to handle each task. The first two moments of such distribution will be then integrated in classic M/G/1 formulas to obtain estimates on the performance indices of interest, e.g., mean waiting time.

Let us first consider an example with two processors, assuming that tasks always split in 10 jobs. We show in Figure 2 the continuous-time Markov chain whose hitting time from initial state (5, 5) to absorbing state (0, 0) represents the service time of one incoming task. In the figure, we omitted some states to better illustrate the thick (diagonal) transitions which correspond to steals. Stealing of jobs only happens on the states at the boundary of the diagram. Considering the general case $R \geq 2$ and job allocation $\mathbf{n} \in \{1, \dots, N_{\max}\}^R$ where exists $s : n_s = 0$, according to Assumptions 1 and 2 we note that a steal removes half of the jobs from the queue of processor $s' = \min\{r : n_r = \max_s n_s\}$ and can be performed only by processor $r' = \min\{s : n_s = 0 \wedge s > s'\}$ if it exists and otherwise by $r' = \min\{s : n_s = 0 \wedge s < s'\}$. Therefore, from state \mathbf{n} of the service time state diagram, a stealing action moves to state

$$\mathbf{n}^* = \mathbf{n} + \lfloor 0.5 \max_s n_s \rfloor \mathbf{e}_{r'} - \lfloor 0.5 \max_s n_s \rfloor \mathbf{e}_{\min\{r : n_r = \max_s n_s\}} \quad (2)$$

where r' is the processor that steals from s' and \mathbf{e}_r is the unit vector in direction r , which means that jobs are stolen from the processor having the largest backlog. In Table 1, we summarize the transition rates of the generalization of the Markov chain depicted in Figure 2.

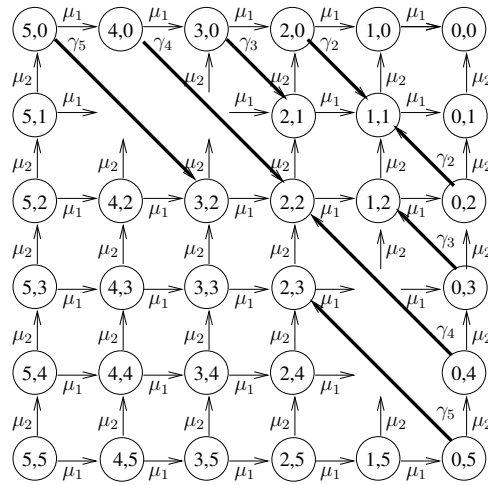


Figure 2: The reducible Markov chain of the task service time distribution with $K = 1$ and $N_K = 5$.

Condition on \mathbf{n}	State transition	Rate
1) $n_r \geq 1, \forall r$	$\forall r : \quad \mathbf{n} \mapsto \mathbf{n} - \mathbf{e}_r$	μ_r
2) $\exists r : n_r = 0 \wedge \exists s : n_s > 1$	$\mathbf{n} \mapsto \mathbf{n}^*$ $\forall t : n_t > 0 : \quad \mathbf{n} \mapsto \mathbf{n} - \mathbf{e}_t$	$\gamma_{\max_t n_t}$ μ_t
3) $n_r \leq 1, \forall r \wedge \exists s : n_s = 0$	$\forall r : n_r = 1 \quad \mathbf{n} \mapsto \mathbf{n} - \mathbf{e}_r$	μ_r

Table 1: Transition rates of the Markov chain characterizing the task service time distribution; \mathbf{e}_r is the unit vector in direction r .

Let $\mathcal{T}_{\mathbf{n}}$ denote the random variable of the task service time in job allocation \mathbf{n} , i.e., when n_r jobs are assigned to processor r , $\forall r$, $T_{\mathbf{n}} := \mathbb{E}[\mathcal{T}_{\mathbf{n}}]$ and $V_{\mathbf{n}} := \mathbb{E}[\mathcal{T}_{\mathbf{n}}^2]$. The (Markovian) representation of the task service time shown in Figures 2 and 1 can be used to derive recursive formulas for the first two moments of $\mathcal{T}_{\mathbf{n}}$.

The following theorem provides recursive formulas for $T_{\mathbf{n}}$ where, for simplicity, we denote

$$\mu = \sum_r \mu_r, \quad \mu_{0,\mathbf{n}} = \gamma_{\max_s n_s} + \sum_{s:n_s>0} \mu_s. \quad (3)$$

Theorem 1 *For all jobs allocations \mathbf{n} , the following relations hold true. If $n_r > 1, \forall r$, then*

$$T_{\mathbf{n}} = \frac{1}{\mu} + \sum_{r=1}^R \frac{\mu_r}{\mu} T_{\mathbf{n}-\mathbf{e}_r}. \quad (4)$$

If $\exists r, s : n_r = 0 \wedge n_s > 1$, then

$$T_{\mathbf{n}} = \frac{1}{\mu_{0,\mathbf{n}}} + \frac{\gamma_{\max_s n_s}}{\mu_{0,\mathbf{n}}} T_{\mathbf{n}^*} + \sum_{r:n_r>0} \frac{\mu_r}{\mu_{0,\mathbf{n}}} T_{\mathbf{n}-\mathbf{e}_r}. \quad (5)$$

If $n_r \leq 1, \forall r$, then

$$T_{\mathbf{n}} = \frac{1}{\sum_{s:n_s=1} \mu_s} + \sum_{r:n_r=1} \frac{\mu_r}{\sum_{s:n_s=1} \mu_s} T_{\mathbf{n}-\mathbf{e}_r} \quad (6)$$

where \mathbf{n}^* is given by (2).

Proof: The above formulas are obtained by applying standard one-step analysis and taking into account the transition rates in Figure 1 of the Markov chain characterizing the task service time distribution. \square

The rationale behind Formula (4) is intuitive: in fact, considering for simplicity $R = 2$ and Figure 2, the mean flow time from state (n_1, n_2) to $(0, 0)$ is given by the mean time spent in state (n_1, n_2) , i.e., $1/\mu$, plus the mean flow times from states $(n_1 - 1, n_2)$ and $(n_1, n_2 - 1)$ multiplied, respectively, by the probabilities of joining these states from (n_1, n_2) , i.e., μ_1/μ and μ_2/μ . On the other hand, when $n_2 = 0$ and $n_1 > 1$, with probability μ_1/μ a transition occurs to state $(n_1 - 1, 1)$ (with no stealing), and with probability $\gamma_{n_1}/(\gamma_{n_1} + \mu_1)$ processor 2 steals $\lfloor n_1/2 \rfloor$ jobs from the local queue of processor 1. The resulting mean service time is given by $1/(\gamma_{n_1} + \mu_1)$ plus the mean service times of both states multiplied by their joining probabilities. When $\mathbf{n} = (1, 1)$, each processor handles exactly one job and, thus, $T_{1,1}$ represents the mean of the maximum of their processing. We notice that even though the mean of the maximum of exponential random variables with different means can be analytically explicit in a non-recursive way, in the general case of heterogeneous processors it does not provide a more efficient computational scheme.

The following theorem analogously provides recursive formulas for the second moment $V_{\mathbf{n}}$.

Theorem 2 *For all jobs allocations \mathbf{n} , the following relations hold true. If $n_r > 1, \forall r$, then*

$$V_{\mathbf{n}} = \frac{2}{\mu^2} + \sum_{r=1}^R \frac{\mu_r}{\mu} \left(V_{\mathbf{n}-\mathbf{e}_r} + 2 \frac{T_{\mathbf{n}-\mathbf{e}_r}}{\mu} \right) \quad (7)$$

If $\exists r, s : n_r = 0 \wedge n_s > 1$, then

$$V_{\mathbf{n}} = \frac{2}{\mu_{0,\mathbf{n}}^2} + \frac{\gamma_{\max_s n_s}}{\mu_{0,\mathbf{n}}} \left(V_{\mathbf{n}^*} + 2 \frac{T_{\mathbf{n}^*}}{\mu_{0,\mathbf{n}}} \right) + \sum_{r:n_r>0} \frac{\mu_r}{\mu_{0,\mathbf{n}}} \left(V_{\mathbf{n}-\mathbf{e}_r} + 2 \frac{T_{\mathbf{n}-\mathbf{e}_r}}{\mu_{0,\mathbf{n}}} \right) \quad (8)$$

If $n_r \leq 1, \forall r$, then

$$V_{\mathbf{n}} = \frac{2}{\left(\sum_{s:n_s=1} \mu_s \right)^2} + \sum_{r:n_r=1} \frac{\mu_r}{\sum_{s:n_s=1} \mu_s} \left(V_{\mathbf{n}-\mathbf{e}_r} + 2 \frac{T_{\mathbf{n}-\mathbf{e}_r}}{\sum_{s:n_s=1} \mu_s} \right) \quad (9)$$

where $T_{\mathbf{n}-\mathbf{e}_r}$ is given by Theorem 1 and \mathbf{n}^* is given by (2).

Proof: The first case algebraically follows by observing that $V_{\mathbf{n}} = \sum_{r=1}^R \frac{\mu_r}{\mu} \mathbb{E}[(\mathcal{T}_{\mathbf{n}-\mathbf{e}_r} + X)^2]$, where X is an exponential random variable with mean μ^{-1} . In the second case, we have $V_{\mathbf{n}} = \sum_{r:n_r>0} \frac{\mu_r}{\mu_{0,\mathbf{n}}} \mathbb{E}[(\mathcal{T}_{\mathbf{n}-\mathbf{e}_r} + Y)^2] + \frac{\gamma_{\max_s n_s}}{\mu_{0,\mathbf{n}}} \mathbb{E}[(\mathcal{T}_{\mathbf{n}^*} + Y)^2]$ where Y is a random variable representing the sojourn time spent in states \mathbf{n} such that $\exists r : n_r = 0 \wedge \exists s : n_s > 1$. For the Markovian assumptions of our model, it is known that $Y_{\mathbf{n}}$ is exponentially distributed with mean $1/(\gamma_{\max_s n_s} + \sum_{s:n_s>0} \mu_s)$. The third case follows analogously. \square

The rationale behind the formulas for $V_{\mathbf{n}}$ is the same discussed above for $T_{\mathbf{n}}$.

3.2 Performance Indices

We now explicit performance indices formulas of the proposed work-stealing model which are expressed in terms of the results of Theorem 1 and 2. Since tasks split into different numbers of jobs, namely $N_k R$ with probability p_k , the mean service time of incoming tasks T is simply obtained by averaging over all possible splits. Assuming, for simplicity, that jobs are equally distributed among processors, we obtain

$$T = \mathbb{E}[\sum_{k=1}^K p_k \mathcal{T}_{\mathbf{N}_k}] = \sum_{k=1}^K p_k T_{\mathbf{N}_k}. \quad (10)$$

Analogously, the second moment is given by

$$\begin{aligned} V &= \mathbb{E}[(\sum_{k=1}^K p_k \mathcal{T}_{\mathbf{N}_k})^2] \\ &= \sum_{k=1}^K p_k^2 \mathbb{E}[\mathcal{T}_{\mathbf{N}_k}^2] + \sum_{\substack{i,j=1 \\ i \neq j}}^K p_i p_j \mathbb{E}[\mathcal{T}_{\mathbf{N}_i} \mathcal{T}_{\mathbf{N}_j}] \\ &= \sum_{k=1}^K p_k^2 V_{\mathbf{N}_k} + \sum_{\substack{i,j=1 \\ i \neq j}}^K p_i p_j T_{\mathbf{N}_i} T_{\mathbf{N}_j}, \end{aligned} \quad (11)$$

where the last step follows by the independence of tasks. Notice that these formulae can be easily extended to the case where jobs do not evenly split among the processors. The mean waiting time W is then given by Pollaczek–Khintchine formula, e.g., [9], yielding

$$W = \frac{\lambda V}{2(1 - \lambda T)}, \quad (12)$$

the mean response time is $W + T$, and the mean number of tasks in the system follows by Little’s law [9].

3.3 Computational Complexity and Comparison with Global Balance

In this section, we analyze the computational cost of our approach in terms of model input parameters and make a comparison with a classic technique. It is clear that the critical issue is the computation of (10) and (11). Let $N_{\max} = \max_{k=1,\dots,K} N_k$. Since $T_{N_{\max},\dots,N_{\max}}$ requires the computation of T_{N_k,\dots,N_k} , for all k , the direct computation of T through (10) and (4) has a complexity equal to the one needed by $T_{N_{\max},\dots,N_{\max}}$. Assuming that one can iterate over set $\Omega(i) := \{\mathbf{n} : \sum_r n_r = i, 0 \leq n_r \leq N_{\max}\}$ in $O(\|\Omega(i)\|)$ steps (by means, e.g., of recursive calls), the computational requirements of the proposed analysis become $O(RN_{\max}^R)$ for time, and $O(N_{\max}^{R-1})$ for space. The former follows from the fact that we need to (diagonally) span each possible job allocation and for each of them perform $O(R)$ elementary operations, and the latter follows from the fact that we need to store the value of each state which can be reached by a steal (see Figure 2). Once T is known, V is obtained at the same computational cost.

Given that the classic global balance technique (see, e.g., [9]) can be applied to our model to obtain its exact (stationary) solution, we now characterize its computational complexity in order to assess the benefit of our approach. Let (m, \mathbf{n}) be a state of the proposed work-stealing model as in (1) where $m \geq 0$ and $0 \leq n_r \leq N_{\max} = \max_{k=1,\dots,K} N_k$. To make global balance feasible and perform the comparison, we consider a state space truncation of process (1) which limits to M the number of tasks in the system. For a given λ , it is known that such truncation yields nearly exact results if M is sufficiently large (note that M should be much larger than RN_{\max}). Since the global balance method is based on the numerical solution of linear equations stating that the probability “flux” outgoing each state must be equal to the probability “flux” going into that state, the cardinality of the resulting system used to obtain the values of mean performance indices is given by the

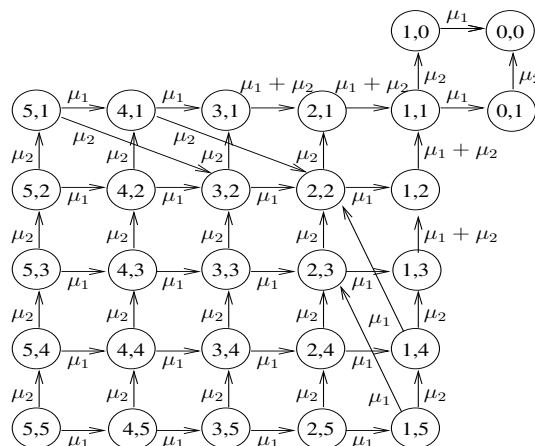


Figure 3: The reducible Markov chain of the task service time distribution when $K = 1$, $N_K = 5$ and $\gamma_n \rightarrow \infty$, $\forall n$.

cardinality of set $\{(m, \mathbf{n}) : 0 \leq m \leq M \wedge 0 \leq n_r \leq N_{\max}, \forall r\}$. Hence, the resulting complexity is given by the computational requirement of the solution of a linear system composed of $O(MN_{\max}^R)$ equations. It is known that this is orders of magnitude more difficult than computing mean performance indices through our approach.

4 Bounding Analysis

Even though the analytical framework introduced in previous section provides the exact solution of our work-stealing model with a computational cost much lower than the one of standard global balance, it remains computationally impractical when tasks split in many jobs or when systems with many processors are considered. In this section we propose an approximation of the task service time distribution which provides efficient bounds on both T and V , and, as a consequence, on the mean task waiting time W and the mean number of tasks.

The proposed bounds are obtained by assuming that the communication delay for transferring jobs among the processors tends to zero, i.e., $\gamma_i \rightarrow \infty$, $\forall i$. This assumption is motivated by the fact that the communication delay is often much smaller than the service time of each job (multiprocessor systems are usually interconnected by very fast buses). Also, note that the transfer of jobs among processor queues can only require to copy the memory address of each job in the local cache of the processor making the steal. Therefore, it is the time needed to read few bytes, which is usually much lower than job service times. In the following, all variables related to the case where $\gamma_i = \infty$ will be denoted with the superindex L .

Consider the two-processor case and, thus, the state diagram of Figure 2. With respect to states of the form $(n_1, 0)$, we observe that if $\gamma_{n_1} \rightarrow \infty$, then with probability 1 the next state becomes $(\lceil n_1/2 \rceil, \lfloor n_1/2 \rfloor)$ and the sojourn time in state $(n_1, 0)$ tends to zero so that these states become vanishing states. Therefore, the Markov chain shown in Figure 2 reduces to the Markov chain depicted in Figure 4.

Theorem 3 *Under the foregoing assumptions, for all \mathbf{n} , $T_{\mathbf{n}}^L \leq_{st} T_{\mathbf{n}}$. This implies $T_{\mathbf{n}}^L \leq T_{\mathbf{n}}$ and $V_{\mathbf{n}}^L \leq V_{\mathbf{n}}$.*

Proof: The proof holds by using a coupling argument between the two Markov chains $\mathbf{n}(t)$ and $\mathbf{n}^L(t)$. Let us uniformize both chains using the uniformizing constant $U = \sum \mu_r + \gamma_{\max}$. The coupled uniformized chains are denoted $\tilde{\mathbf{n}}(k)(\omega)$ and $\tilde{\mathbf{n}}^L(k)(\omega)$ and are constructed over the same finite probability space $(\Omega, P) = \{(0, \gamma/U), (1, \mu_1/U), \dots, (R, \mu_R/U)\}$.

For the original Markov case, if $\tilde{\mathbf{n}}(k)(\omega) > (0, \dots, 0)$, then

- if $\omega = i$ then $\tilde{\mathbf{n}}(k+1)(\omega) = \tilde{\mathbf{n}}(k)(\omega) - \mathbf{1}_i$,
- if $\omega = 0$ then the state is unchanged.

However, if $\tilde{\mathbf{n}}(k)(\omega)_i = 0$ and $\omega = 0$, then a steal transition takes place, and if $\omega = i$ the state is unchanged.

As for the modified case, if $\tilde{\mathbf{n}}^L(k)(\omega) > (1, \dots, 1)$ then

- if $\omega = i$ then $\tilde{\mathbf{n}}^L(k+1)(\omega) = \tilde{\mathbf{n}}^L(k)(\omega) - \mathbf{1}_i$,

- if $\omega = 0$ then the state is unchanged.

However, if $\tilde{\mathbf{n}}^L(k)(\omega)_i = 1$ and $\omega = i$, then a service in queue i and a steal take place together, and if $\omega = 0$ the state is unchanged.

By considering both chains and as long as the sum $\tilde{n}_1^L(k)(\omega) + \dots + \tilde{n}_r^L(k)(\omega)$ is larger than 1, $\tilde{n}_1^L(k)(\omega) + \dots + \tilde{n}_r^L(k)(\omega)$ decreases by 1 whenever $\omega \neq 0$ and remains unchanged when $\omega = 0$, while $\tilde{n}_1^L(k)(\omega) + \dots + \tilde{n}_r^L(k)(\omega)$ also remains unchanged when $\omega = 0$, but also in certain cases when $\omega \in \{1, \dots, r\}$. This implies (by a straightforward induction on k) that $\tilde{n}_1^L(k)(\omega) + \dots + \tilde{n}_r^L(k)(\omega) \leq \tilde{n}_1(k)(\omega) + \dots + \tilde{n}_r(k)(\omega)$ for all integer k . Therefore, for the original continuous time chains, for all time t , $n_1^L(t)(\omega) + \dots + n_r^L(t)(\omega) \leq_{st} n_1(t)(\omega) + \dots + n_r(t)(\omega)$ where \leq_{st} is the stochastic order [15]. Finally, since the random \mathcal{T}_n is a non-decreasing function of the sums, $\mathcal{T}_n^L \leq_{st} \mathcal{T}_n$. \square

In the transition matrix of this new Markov chain, we observe that the sojourn times of each state (n_1, n_2) such that $n_1, n_2 \geq 1$ are i.i.d. random variables exponentially distributed with mean $1/\mu$. Since any path from initial state (N, N) to absorbing state $(1, 1)$ involves $2N - 2$ steps, we conclude that the distribution of the time needed to reach state $(1, 1)$ from (N, N) is Erlang with rate parameter μ and $2N - 2$ phases. Including the sojourn times of states $(1, 1)$, $(1, 0)$ and $(0, 1)$, the task service time distribution becomes $\mathcal{T}_{N,N}^L \stackrel{db}{=} \text{Erlang}(\mu, 2N - 2) + \max\{X_1, X_2\}$, where X_r denotes an exponential random variable with mean μ_r^{-1} . It is easy to see that this observation holds even in the more general case where $R \geq 2$ and tasks can split in different numbers of jobs. The resulting task service time distribution becomes (when $\gamma_i \rightarrow \infty$)

$$\mathcal{T}^L \stackrel{db}{=} \sum_{k=1}^K p_k \mathcal{T}_{\mathbf{N}_k}^L \stackrel{db}{=} \sum_{k=1}^K p_k \left(\text{Erlang}(\mu, N_k R - R) + \max_{r=1, \dots, R} \{X_r\} \right). \quad (13)$$

Therefore, as shown for (11),

$$T^L = T_1 + \sum_{k=1}^K p_k \frac{N_k R - R}{\mu}, \quad V^L = \sum_{k=1}^K p_k^2 V_{\mathbf{N}_k}^L + \sum_{\substack{i,j=1 \\ i \neq j}}^K p_i p_j T_{\mathbf{N}_i}^L T_{\mathbf{N}_j}^L, \quad (14)$$

where

$$V_{\mathbf{N}_k}^L = \mathbb{E}[(\mathcal{T}_{\mathbf{N}_k}^L)^2] = \frac{N_k R - R + (N_k R - R)^2}{\mu^2} + V_1 + 2 \frac{N_k R - R}{\mu} T_1 \quad (15)$$

$k = 1, \dots, K$, are lower bounds on the first two moments of \mathcal{T} by means of Theorem 3. In turn, the mean waiting W straightforwardly becomes a lower bound by means of (12).

In (14), the computational complexity of T^L and V^L is then dominated by the computation of T_1 (note that V_1 is obtained at the same computational cost as T_1). By means of Formula (6), this is given by $O(R2^R + K)$ for time and $O(R + K)$ for space. Therefore, the computational complexity of the proposed bounds becomes independent of N_{\max} . Even though our bounding analysis has a complexity which is exponential in the number of processors, we observe that multiprocessor embedded systems are usually composed of a limited number of processors. In our context, this makes our bounds efficient.

4.1 Homogeneous Processors

In many cases of practical interest, multiprocessor systems are composed of identical processors. In our model, this implies $\mu_1 = \dots = \mu_R$. In this particular case, we observe that very efficient expressions can be derived for T_1 and V_1 . Noting that T_1 is the maximum of R i.i.d. exponential random variables, it is a well-known result of extreme-value statistics, e.g., [11], that

$$T_1 = \mu_1^{-1} \sum_{r=1}^R r^{-1}$$

and

$$V_1 = \mu_1^{-2} \sum_{r=1}^R r^{-2} + T_1^2,$$

which are computationally much more efficient than Formulae (6) and (9).

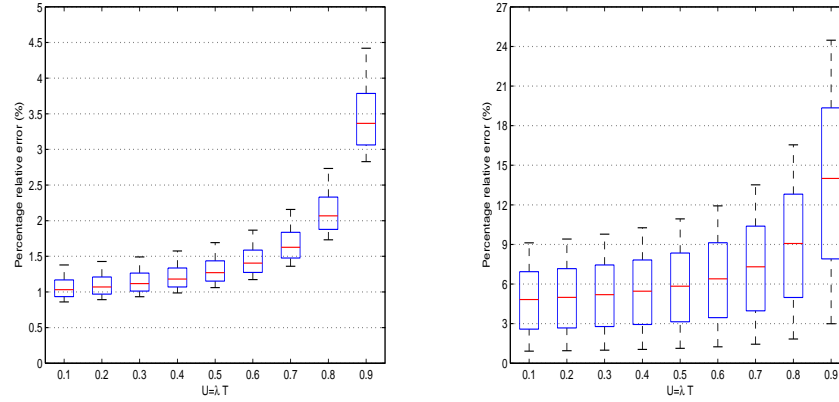


Figure 4: Boxplot of errors (16) in the cases of homogeneous (on the left) and heterogeneous (on the right) processors.

5 Numerical Results

In this section, we numerically assess the accuracy of our approach. Numerical results are presented relative to three different sets of experiments. First, we perform a validation of the proposed bounds on a wide test-bed of randomly generated models to assess the general quality of the relative error percentages. In this setting, we distinguish between the homogeneous and heterogeneous case. Then, we show asymptotic performances of our bounds when N_{\max} is large.

To assess the general accuracy of the proposed bound with respect to the exact model solution, we consider a test bed of 3,000 randomly-generated models, and for each test we compute

$$100\% \cdot (W_{exact} - W_{bound}) / W_{exact}, \quad (16)$$

i.e., the percentage relative error on mean waiting time. Instead of considering the errors of T_{bound} and V_{bound} , we directly consider the error (16) because it is much less robust than the formers by means of (12). Exact model solutions have been obtained through the analysis presented in Section 3. The input parameters used to validate the proposed bounds are shown in Table 2. Since real-world embedded systems are composed of a

	Interval		Value
Number of processors (R)	{2, 3, 4}	Distribution of task splits (p_k)	$1/K$
Jobs mean service times (μ_r)	[0.1, 10]	Number of jobs per task of type k (N_k)	$k \cdot 20$
Number of possible task splits (K)	{2, ..., 10}	Communication delay (γ_i^{-1})	$\lfloor \frac{i}{2} \rfloor / (10 \frac{\mu}{R})$

Table 2: Parameters used in the validation of the proposed bound.

limited number of processors, in our tests we consider $R \leq 4$. We did not consider tests with a larger size of $\max_{k=1, \dots, K} N_k$ because of the computational requirements needed to obtain the exact solution and the consequent cost of computing robust results for the large number of test cases. Within the parameters shown in Table 2, note that the maximum number of jobs which can be assigned to a processor is 200. The communication delay γ_i^{-1} is assumed to be linear in the number of task to transfer and we also assume that the time needed to transfer one job is ten times lower than the mean service time of that job. We now perform a parametric analysis by varying the mean task arrival rate λ such that the mean system utilization, i.e., $U = \lambda T$ (see, e.g., [14]), range between 0.1 (light-load) and 0.9 (heavy-load). We first conduct a numerical analysis assuming that the processors are homogeneous, i.e., $\mu_1 = \dots = \mu_R$. In Figure 4 (on the left) we illustrate the quality of the error (16) by means of the Matlab *boxplot* command, where each box is referred to an average of 3,000 models. In this figure, our bounds provide very accurate results with an average error less than 2%. As the system utilization U increases, a slight loss of accuracy occurs due to the analytical expression of Formula (12) which makes the mean waiting time W very sensitive to small errors on T as U grows to unity. However, in the worst-case where $U = 0.9$, our bound provides again a very small average error, i.e., 3.4%. Also, our bounds are robust because the variance of the waiting-time errors is very small.

We now focus on the quality of error (16) within the same setting above but in the heterogeneous case, i.e., assuming that all processors are identical but one, which is faster than the other ones. We assume that the *speed-up*

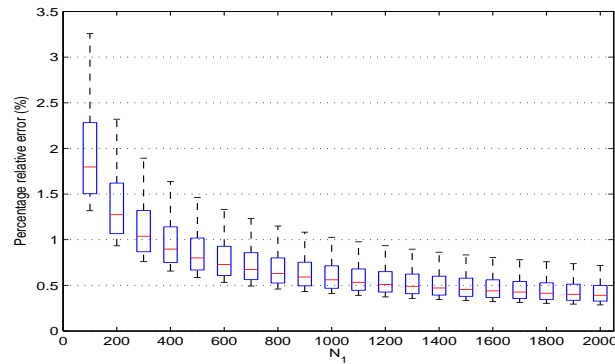


Figure 5: Boxplot of errors (16) when tasks split in a large number of jobs.

of the fastest processor, i.e., the ratio between its mean service rate and the one of the other processors, is a random variable uniformly distributed in range $(1, 2]$ (in real-world embedded systems, the typical speed ratio is below 1.5). Figure 4 (on the right) displays the error (16), where each box refers to the average over 3,000 models. Again, our bounds are accurate even though there is a slight loss of accuracy with respect to previous cases. This could be due to the fact that the fastest processor tends to perform more steals than in the homogeneous case so that the overall communication delay becomes less negligible if compared to the mean task service time. However, the average error remains small and it assesses to 7%. The fact that the error (16) is increasing in U finds the same explanation discussed above for the homogeneous case. Since the variance of the waiting-time errors remains limited, our results remain robust.

Now, consider that tasks split in a very large number of jobs (often the case of many real-world applications: for instance, this is the case where tasks are images and jobs execute algorithms on (small) groups of pixels). Due to the expensive computational effort of calculating the exact model solution, we limit such analysis to two-processor systems, i.e., $R = 2$. Within the input parameters shown in Table 2, we perform a parametric analysis by increasing the mean number of jobs per task. We consider homogeneous processors and assume $K = 1$, which means that tasks always split in $N := RN_1$ jobs, where N_1 varies from 100 to 2,000 with step 50, i.e., a task can split in 4,000 jobs at most. To better stress the accuracy of the bounds, we consider the worst case where the input arrival rate λ is such that $U = \lambda T$ ranges in $[0.7, 0.9]$ (see Figure 4). In Figure 5 we show the quality of error (16) with the Matlab boxplot command, where each box is referred to 1,000 models. Our bounds yield nearly exact results when tasks split in a large number of jobs and the average error decreases as N_1 increases. When $N_1 \geq 400$, the average error becomes lower than 1%. Within the large number of test cases, the proposed bounds are robust because also the variance of the average errors is small and decreases as N_1 increases.

6 Optimal Number of Processors

In this section, we show how the proposed analysis can be applied in the context of optimization. Here, the objective is to minimize infrastructural costs (energy consumption), determined by both the speed and the number of processors, while satisfying constraints on waiting times. We assume that the task arrival rate λ is given and that the mean waiting time of tasks must not exceed \bar{W} units of time. Our optimization model applies at *run-time*, and must be re-executed whenever λ (or \bar{W}) changes to determine the new optimal number of active processors and their speed. The latters can be adjusted by means of *frequency scaling* threads. We also assume the case of homogeneous processors because it often happens in practice. Therefore, the optimization is obtained by solving the following mathematical program

$$\min Rc(\mu_1), \quad \text{subject to: } W(\mu_1, R) \leq \bar{W}, \quad \mu_1 \in \mathbb{R}^+, \quad R \in \mathbb{N}, \quad (17)$$

where $c(\mu_1)$ is the *cost* of using a single processor having processing speed μ_1 . If the cost corresponds to the instantaneous power consumption, then for each processor, the cost can be approximated by $c(\mu_1) = A\mu_1^\alpha$, where A is a constant and $\alpha \geq 1$, typically $2 \leq \alpha \leq 3$ for most circuit models (see e.g., [12]). The solution of (17) provides the optimal speed and number of processors w.r.t. energy use, in order to satisfy the performance requirement. Since operating speeds of processors can be controlled by power management threads, in our model these are

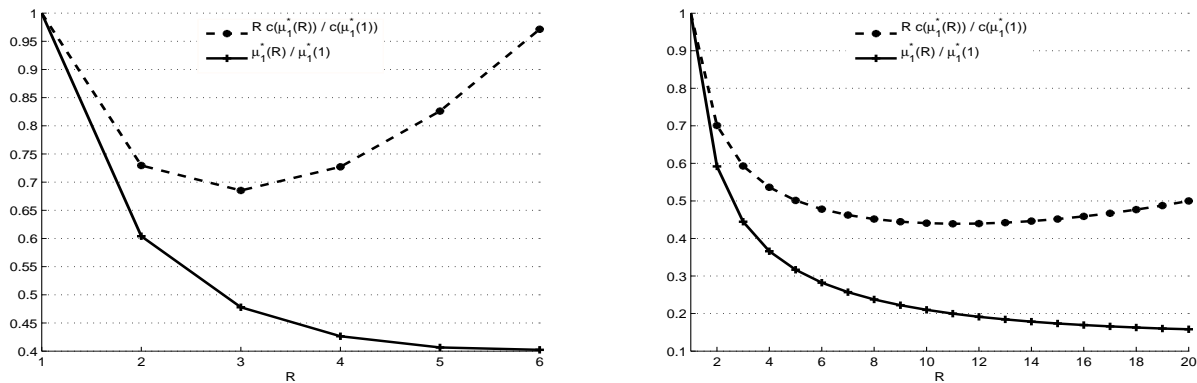


Figure 6: Cost benefits of the work-stealing algorithm with respect to the optimal single-processor configuration when $\lambda = 1$ jobs per unit of time, $W = 1$, $K = 1$ (on the left) and $K = 10$ (on the right), each task generates $N_k = 100k$ jobs with probability $1/K$, and the cost parameter is $\alpha = 2$.

assumed to be positive real numbers. Since the exact solution of (17) is computationally difficult we exploit the bounds shown in previous sections to obtain an approximate solution in closed form. In this homogeneous case, our bounds are very tight (see Section 5) so that the following does not really depend on work stealing but rather on some form of an ideal parallelism. Noting that with a fixed R , say \bar{R} , both $c(\mu_1)$ and $W(\mu_1, \bar{R})$ are convex and, respectively, monotonically increasing and decreasing in μ_1 , the structure of program (17) ensures that the optimum is achieved when $W(\mu_1, \bar{R}) = \bar{W}$. Adopting formulas (14), this yields a polynomial with degree two and with only one positive root:

$$\mu_1^*(\bar{R}) = \frac{1}{2\bar{R}} \left(\lambda T(1) + \sqrt{(\lambda T(1))^2 + 2\lambda V(1)/\bar{W}} \right) \quad (18)$$

where $T(1)$ and $V(1)$ are given by (14) with $R = \bar{R}$ and $\mu_r = 1/\bar{R}$, $\forall r$. For R fixed, Equation (18) explicits the dependence between \bar{W} and the optimal processing rate: as \bar{W} decreases (being positive), μ_1 must increase with the power of a square root. This immediately shows the benefit of work-stealing with respect to, for instance, a “no-steal” policy, which makes μ_1 increases linearly as \bar{W} decreases (this can be checked by means of standard M/M/1 formulas). Also note that the optimal speed of the processor does not depend on the parameter α so that it is insensitive to the exact model of energy consumption (we only use the fact that this energy use is convex in the processor speed).

To determine the global optimum of (17), we iterate (18) over R . Within parameters of practical interest, in Figure 6 we plot the values of $Rc(\mu_1^*(R))/c(\mu_1^*(1))$ and $\mu_1^*(R)/\mu_1^*(1)$, by varying R only from 1 to 6 (we remark that R is small in the context of embedded systems). These functions represent, respectively, the benefit, in terms of costs, of adopting R processors operating with the work-stealing algorithm with respect to the optimal single-processor configuration, and how much the rate of service of each processor varies (as R varies) to guarantee the waiting time requirement in (17). We consider two scenarios: on the left (on the right), we assume that tasks split in a relatively small (large) number of jobs. In any case, we impose $\bar{W} = \lambda^{-1} = 1$ time unit because embedded systems are usually aimed to perform on-the-fly real-time operations. Within these parameters, Little’s law [9] ensures that the mean number of waiting tasks is one. In the figures, we see that work-stealing yields a remarkable improvement in terms of costs even when $R = 2$, for which a cost reduction of nearly 30% is achieved in both cases. This is obtained with two (identical) processors having speeds reduced of a factor of nearly 1.7. In the first scenario, we observe that the optimum is achieved with $R = 3$ processors. For $R > 3$, the R term in the objective function becomes non-negligible. In the second scenario, a much higher number of processors is needed to make the objective function increase because tasks generate a much larger workload. In fact, to guarantee the waiting time constraint, in this case processors must have a much higher service rate than the corresponding ones of the previous case, and this impacts significantly on the square term μ_1^α of the objective function. In this case, the optimum is achieved with $R = 12$ processors, and even when $R = 2$, work stealing yields a non-negligible cost reduction.

7 Adapting the Fraction of Jobs to Steal

In previous sections, we analyzed the performance of the work-stealing algorithm which steals half of the jobs from some processor queue. However, other stealing functions can be considered, and the proposed framework lets us evaluate their impact by slightly adapting the formulas in Theorems 1 and 2. We now numerically show that some gains can be obtained by adapting the amount of jobs stolen to processor speeds.

Considering job allocation \mathbf{n} and assuming that processor s is the most loaded one, one could consider the following stealing functions which balance the mean load among processors: i) *Adaptive Local (L)*: an idle processor (say r) steals $\alpha(\mathbf{n}) = n_s \cdot \mu_r / (\mu_r + \mu_s)$ jobs from s , and ii) *Adaptive Global (G)*: an idle processor (say r) steals $\alpha(\mathbf{n}) = n_s \cdot \mu_r / \mu$ jobs from s . The concept behind L is to steal the weighted fraction of jobs which balances the mean load between the processor which steals and the one which is stolen. In this manner, after a steal different processors tend to finish the execution of their jobs at nearly the same time with no further steals, and, thus, the overall communication cost is essentially reduced. Similarly, the policy G steals the weighted fraction of jobs which balances the mean global mean load between all processors. We now numerically evaluate the two strategies above within three different scenarios. As in previous sections, we first assume that stealing costs are linear in the number of jobs to transfer. Secondly, we consider the case where the code of the jobs to steal (in the FIFO queue of some processor) have consecutive memory addresses, so that a logarithmic (in $\alpha(\mathbf{n})$) amount of data can be transferred, i.e., the memory address of the initial job and the number of jobs to transfer. Finally, we consider the ideal case where the stealing cost is a constant. In the linear case we assume $\gamma_{\alpha(\mathbf{n})}^{-1} = \alpha(\mathbf{n}) / (10 \frac{\mu}{R})$ (see Section 5), in the logarithmic case we assume $\gamma_{\alpha(\mathbf{n})}^{-1} = \log_2(\alpha(\mathbf{n})) / (10 \frac{\mu}{R})$, and in the constant case we assume $\gamma_{\alpha(\mathbf{n})}^{-1} = 1$. Within the parameters of Table 2 and the settings above, in Table 3 we show the percentage relative errors $\text{Err}_i^1 = 100 \cdot (C_i - C_{1/2}) / C_{1/2}$ and $\text{Err}_i^2 = 100 \cdot (U_i - U_{1/2}) / U_{1/2}$, averaged over 2,000 random models, where C_x and U_x are respectively the first and second moment of the total time spent for stealing where $x \in \{L, G, 1/2\}$ refers to the stealing strategy. In the linear case, strategy G is better (on average) than both L and the strategy

	Linear		Logarithmic		Constant	
	Err_i^1	Err_i^2	Err_i^1	Err_i^2	Err_i^1	Err_i^2
$i = L$	10.30%	10.37%	-3.64%	-3.53%	-3.4%	-3.43%
$i = G$	-8.80%	-8.94%	7.14%	7.11%	11.2%	11.46%

Table 3: Numerical evaluation of the stealing strategies L and G with respect to the *classic* one.

stealing halves of jobs (i.e. $i = 1/2$) yielding a 8.8% reduction in both moments of communication cost. However, this is not true in the other cases, where L turns out to be the best. Hence, the goodness of one policy strongly depends on the structure of communication costs and the impact of adopting a different strategy is not immediate to understand without our analytical framework, which is well-suited to evaluate new strategies. In any case, we notice that $C_{1/2}$ and $U_{1/2}$ tend to be between the corresponding values for the strategies L and G .

Within the parameters given in Table 2, in Figure 7 we illustrate the average error on waiting times, i.e., Formula (16), committed by the considered work-stealing algorithm with respect to the two strategies the stealing strategies L and G in the linear and logarithmic cases. Each point refers to an average of 2,000 randomly generated models. In the figure, we observe that the average percentage relative error of the above load-balancing functions is negligible. Therefore, the impact of the strategies above is negligible within our purposes.

8 Conclusions

In this paper, we analyzed the performance of parallel stream computations on a multiprocessor architecture implementing the work-stealing principle. We proposed a Markovian queueing model able to take into account innovative aspects such as bus contentions and task synchronizations. An analytical framework has been developed for obtaining its exact solution with a computational cost much lower than the one of standard techniques. However, since our solution method remains computational impractical when tasks split in many jobs or many processors are considered, we proposed tight bounds able to efficiently solve our model in an approximate manner. We showed an application of our analysis in the context of determining the optimal number of processors which minimizes infrastructural costs while satisfying waiting time constraints. This is obtained through the solution of a simple convex program. We also showed how our framework applies to other stealing strategies. The goodness of

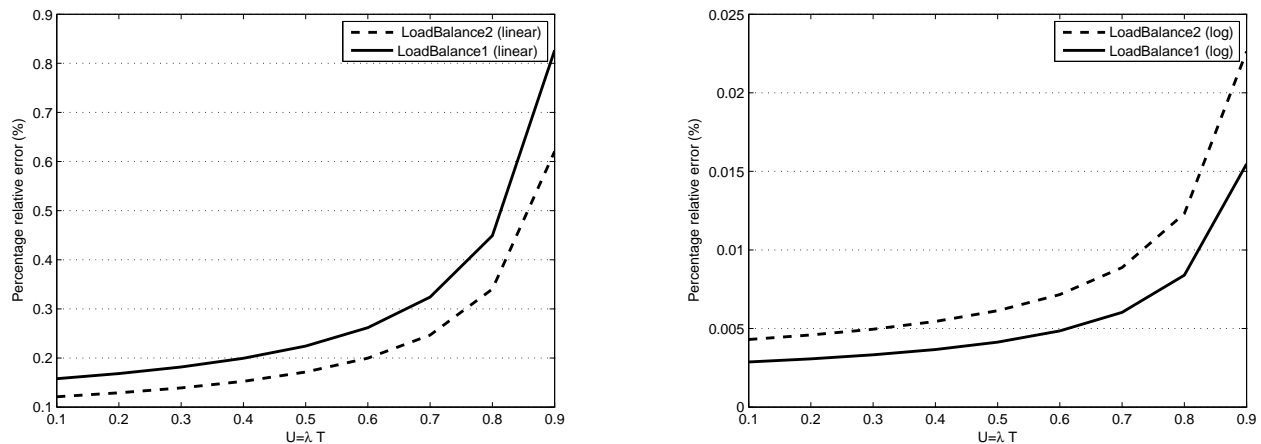


Figure 7: Average percentage relative errors for different stealing functions. On the left (on the right) the cost of stealing is assumed to be linear (logarithmic) in $\alpha(\mathbf{n})$, where $\alpha(\mathbf{n})$ is the number of jobs to steal determined by policies LoadBalance1 and LoadBalance2.

different stealing strategies on task service times has been numerically shown to strongly depend on the structure of communication costs.

Our work addresses future research in several directions. Firstly, we leave as future work the development of more efficient exact and approximate analyses. These include, for instance, the analysis of the mean waiting time in some limiting regime, to obtain bounds or approximations taking into account communication costs. Secondly, the analysis of the task service time distribution when *state-dependent* service rates are considered, and, finally, the modeling of task service times when jobs have dependencies.

Acknowledgements The authors are grateful to Jean-Louis Roch for fruitful discussions on work stealing.

References

- [1] Description of traviata architecture, 2008. <http://www.stlinux.com/drupal/hw/boards/mb426>.
- [2] kaapi software (MOAIS, INRIA project-team), 2008. <http://gforge.inria.fr/projects/kaapi>.
- [3] Umut A. Acar, Guy E. Blelloch, and Robert D. Blumofe. The data locality of work stealing. In *SPAA '00: Proc. of the twelfth annual ACM symposium on Parallel algorithms and architectures*, pages 1–12, New York, NY, USA, 2000. ACM.
- [4] Nimar S. Arora, Robert D. Blumofe, and C. Greg Plaxton. Thread scheduling for multiprogrammed multiprocessors. *Theory Comput. Syst.*, 34(2):115–144, 2001.
- [5] M. A. Bender and M. O. Rabib. Online scheduling of parallel programs on heterogeneous systems with applications to cilk. *Theory of Computing Systems*, 35:289–304, 2002. Special issue on SPA00.
- [6] Petra Berenbrink, Tom Friedetzky, and Leslie Ann Goldberg. The natural work-stealing algorithm is stable. In *In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 178–187, 2001.
- [7] Julien Bernard, Jean-Louis Roch, and Daouda Traore. Processor-oblivious parallel stream computations. In *16th Euromicro International Conference on Parallel, Distributed and network-based Processing*, Toulouse, France, Feb 2008.
- [8] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.
- [9] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor Shridharbhai Trivedi. *Queueing Networks and Markov Chains*. Wiley-Int., 2005.

-
- [10] Matteo Frigo, Charles E. Leiserson, and Keith H. Randall. The implementation of the cilk-5 multithreaded language. In *PLDI '98: Proc. of SIGPLAN 1998 conf. on Progr. lang. design and implementation*, pages 212–223, New York, USA, 1998. ACM.
 - [11] E. J. Gumbel. *Statistics of extremes*. Columbia University Press, New York, 1958.
 - [12] Rabaey J. and Pedram M. *Low Power Design Methodologies*. Kluwer Academic Publishers, 1996.
 - [13] S. Jafar, T. Gautier, A.W. Krings, and J.-L. Roch. A checkpoint/recovery model for heterogeneous dataflow computations using work-stealing. In *Proc. European Conf. Parallel Processing (EuroPar '05)*, pages 675–684, 2005.
 - [14] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Upper Saddle River, NJ, USA, 1984.
 - [15] A. Muller and D. Stoyan. *Comparison methods for stochastic models and risks*. Wiley series in Probability and Statistics, 2002.
 - [16] D. Neill and A. Wierman. On the benefits of work stealing in shared-memory multiprocessors. www.cs.cmu.edu/acw/15740/paper.pdf.
 - [17] Mark S. Squillante and Randolph D. Nelson. Analysis of task migration in shared-memory multiprocessor scheduling. *SIGMETRICS Perform. Eval. Rev.*, 19(1):143–155, 1991.



Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

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