Computing the throughput of probabilistic and replicated streaming applications

Anne Benoit, Matthieu Gallet, Bruno Gaujal, Yves Robert

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Computing the Throughput of Probabilistic and Replicated Streaming Applications

Anne Benoit — Matthieu Gallet — Bruno Gaujal — Yves Robert

N° 7510

January 2011

Distributed and High Performance Computing

ISSN 0249-6399 ISRN INRIA/RR--7510--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

Computing the Throughput of Probabilistic and Replicated Streaming Applications

Anne Benoit , Matthieu Gallet , Bruno Gaujal , Yves Robert

Theme : Distributed and High Performance Computing
Équipes-Projets GRAAL, MESCAL

Rapport de recherche n° 7510 — January 2011 — 33 pages

Abstract: In this paper, we investigate how to compute the throughput of probabilistic and replicated streaming applications. We are given (i) a streaming application whose dependence graph is a linear chain; (ii) a one-to-many mapping of the application onto a fully heterogeneous target, where a processor is assigned at most one application stage, but where a stage can be replicated onto a set of processors; and (iii) a set of I.I.D. (Independent and Identically-Distributed) variables to model each computation and communication time in the mapping. How can we compute the throughput of the application, i.e., the rate at which data sets can be processed? We consider two execution models, the **Strict** model where the actions of each processor are sequentialized, and the **Overlap** model where a processor can compute and communicate in parallel.

The problem is easy when application stages are not replicated, i.e., assigned to a single processor: in that case the throughput is dictated by the critical hardware resource. However, when stages are replicated, i.e., assigned to several processors, the problem becomes surprisingly complicated: even in the deterministic case, the optimal throughput may be lower than the smallest internal resource throughput. To the best of our knowledge, the problem has never been considered in the probabilistic case. The first main contribution of the paper is to provide a general method to compute the throughput when mapping parameters are constant or follow I.I.D. exponential laws. The second main contribution is to provide bounds for the throughput when stage parameters are arbitrary I.I.D. and N.B.U.E. (New Better than Used in Expectation) variables: the throughput is bounded from below by the exponential case and bounded from above by the deterministic case.

Key-words: Scheduling, probabilistic streaming applications, replication, throughput, timed Petri nets..

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Computing the Throughput of Probabilistic and Replicated Streaming Applications

Résumé : Nous nous intéressons dans ce rapport au calcul du débit pour des applications pipelinées probabilites et répliquées.

Mots-clés : Ordonnancement, applications pipelinées probabilistes, réplication, débit, réseaux de Petri.

Contents

1 Introduction

In this paper, we deal with streaming applications, or *workflows*, whose dependence graph is a linear chain composed of several stages. Such applications operate on a collection of data sets that are executed in a pipeline fashion [18,19,23]. They are a popular programming paradigm for streaming applications like video and audio encoding and decoding, DSP applications, etc [10,21,25]. Each data set is input to the linear chain and traverses it until its processing is complete. While the first data sets are still being processed by the last stages of the pipeline, the following ones have started their execution. In steady state, a new data set enters the system every $\mathcal{P}$ time-units, and several data sets are processed concurrently within the system. A key criterion to optimize is the *period*, or equivalently its inverse, the *throughput*. The period $\mathcal{P}$ is defined as the time interval between the completion of two consecutive data sets. The system can process data sets at a rate $\rho = 1/\mathcal{P}$, where ρ is the throughput.

The application is executed on a fully heterogeneous platform, whose processors have different speeds, and whose interconnection links have different bandwidths. We assume that the mappping of the application onto the platform is given, and that this mapping is one-to-many. In other words, when mapping application stages onto processors, we enforce the rule that any given processor will execute at most one stage. However, a given stage may well be executed by several processors. Indeed, if the computations of a given stage are independent from one data set to another, then two consecutive computations (for different data sets) of the same stage can be mapped onto distinct processors. Such a stage is said to be *replicated*, using the terminology of Subhlok and Vondran [19,20] and of the DataCutter team [6,18,24]. This also corresponds to the *dealable* stages of Cole [9]. Finally, we consider two execution models, the **Strict** model where the actions of each processor are sequentialized, and the **Overlap** model where a processor can process a data set while it is simultaneously sending the previous data set to its successor and receiving the next data set.

Unsurprisingly, the problem of finding a mapping with optimal throughput of tasks on processors is NP-complete, even in the deterministic case and without any communication cost [3]. However, even determining the throughput may be difficult. In the deterministic case, and without replication, the throughput of a given mapping is easily seen to be dictated by the critical hardware resource: the period is the largest cycle-time of any resource, be it a processor or communication link. However, when stages are replicated, the problem becomes surprisingly complicated: even in the deterministic case, the optimal throughput may be lower than the smallest internal resource throughput. In this paper, we present a model based on timed event graphs to determine this throughput.

We also introduce randomness in the execution of the application onto the platform. Consider the computations performed by a given processor on different data sets: we assume that the execution times of the computations are random variables that obey arbitrary I.I.D. (Independent and Identically-Distributed) probability laws. Similarly, we assume that the execution times of all the communications taking place on a given interconnection link are random variables that obey arbitrary I.I.D. probability laws. Note that the I.I.D. hypothesis apply to all events (computations or communications) that occur on the same hardware resource (either a processor or a communication link),

and does not restrict the heterogeneity of the application/platform mapping. In other words, processors may well have different speeds, links may well have different bandwidths, stages may well have very different computation and data volumes; furthermore, the distribution law may well vary from one computation to another, or from one communication to another.

The main contribution is to provide a general method (although of exponential cost) to compute the throughput when mapping parameters follow I.I.D. exponential laws. This general method is based upon the detailed analysis of the timed Petri nets deduced from the application mapping for each execution model, **Strict** and **Overlap**. It turns out that the Petri nets exhibit a regular structure in the **Overlap** model, thereby enabling to reduce the cost and provide a polynomial algorithm. The second main contribution is to provide bounds for the throughput when stage parameters are arbitrary I.I.D. and N.B.U.E. (New Better than Used in Expectation) variables: the throughput is bounded from below by the exponential case and bounded from above by the deterministic case.

The paper is organized as follows. First in Section 2, we formally describe the framework and the optimization problems, and we introduce the random variables that are used for the probabilistic study. Then we briefly introduce in Section 3 timed event graphs which are used to solve the deterministic case. Using these event graphs, we solve the deterministic (or static) case in Section 4. We explain how to compute the throughput when communication and computation times follow I.I.D. exponential laws (Section 5). We give a general method which turns out to be of exponential complexity in the general case, but we provide a polynomial algorithm for the **Overlap** model. Then in Section 6, we deal with arbitrary I.I.D. and N.B.U.E. laws, and we establish the above-mentioned bounds on the throughput. Some experimental results are given in 7, both to assess the quality of our model and to observe the behaviour of more complex random laws. Finally, we present some conclusions and directions for future work in Section 8.

2 Models

In this section, we first describe the workflow application, the target platform, and the communication models that we consider (Section 2.1). The replication model is presented in Section 2.2. We formally define the throughput in Section 2.3. Finally, we give a detailed presentation of the random variables that we consider to model processor speeds and link bandwidths (Section 2.4).

2.1 Application, platform and communication models

We deal with streaming applications, or *workflows*, whose dependence graph is a linear chain composed of N stages, called T_i ($1 \leq i \leq N$). Each stage T_i has a size w_i, expressed in flop, and needs an input file F_{i-1} of size δ_{i-1}, expressed in bytes. Finally, T_i produces an output file F_i of size δ_i, which is the input file of stage T_{i+1}. All these sizes are independent of the data set. Note that T_1 produces the initial data and does not receive any input file, while T_N gathers the final data.

The workflow is executed on a fully heterogeneous platform with M processors. The speed of processor P_p ($1 \leq p \leq M$) is denoted as s_p (in flops). We assume bidirectional links $\text{link}_{p,q} : P_p \to P_q$ between any processor pair P_p and P_q, with bandwidth $b_{p,q}$ bytes per second. These links are not necessarily physical, they can be logical. For instance, we can have a physical star-shaped platform, where all processors are linked to each other through a central switch. The time needed to transfer a file F_i from P_p to P_q is $\frac{\delta_i}{b_{p,q}}$, while the time needed to process T_i on P_p is $\frac{w_i}{s_p}$. An example of linear chain application and fully connected target platform is provided in Figure 1.

We consider two different realistic common models for communications. The **Overlap** model allows to overlap communications and computations: a processor can simultaneously receive values for the next data set, compute result for the current data set, and send output data for the previous data set. Requiring multi-threaded programs and full-duplex network interfaces, this model allows for a better use of computational resources. On the contrary, in the **Strict** model, there is no overlap of communications by computations: a processor can either receive a given set of data, compute its result or send this result. This is the typical execution of a single-threaded program, with one-port serialized communications. Although leading to a less efficient use of physical resources, this model allows for simpler programs and hardware.

2.2 Replication model

When mapping application stages onto processors, we enforce the rule that any given processor will execute at most one stage. But instead of considering *one-to-one mappings* [5], we allow stage replication, and rather consider one-to-many mappings, in which each stage can be processed by several processors. This is possible when the computations of a given stage are independent from one data set to another. In this case, two consecutive computations (different data sets) for the same stage can be mapped onto distinct processors. Such a stage is said to be *replicated* [6, 18–20, 24] or *dealable* [9].

Note that the computations of a replicated stage can be fully sequential for a given data set, what matters is that they do not depend upon results from previous data sets, hence the possibility to process different data sets in different locations. The following scheme illustrates the replication of a stage T_i onto three processors:

$$\ldots\ T_{i-1} \begin{cases} \nearrow \\ -- \\ \searrow \end{cases} \begin{array}{l} T_i \text{ on } P_1\text{: data sets } \mathbf{1, 4, 7, \ldots} \\ T_i \text{ on } P_2\text{: data sets } \mathbf{2, 5, 8, \ldots} \\ T_i \text{ on } P_3\text{: data sets } \mathbf{3, 6, 9, \ldots} \end{array} \begin{cases} \searrow \\ -- \\ \nearrow \end{cases} T_{i+1}\ \ldots$$

For $1 \leq i \leq N$, let R_i denote the number of processors participating to the processing of T_i. For $1 \leq p \leq M$, if P_p participates to the work of T_i, then we write $p \in \mathit{Team}_i$ and define $R'_p = R_i$. As outlined in the scheme, the processors allocated to a replicated stage execute successive data sets in a round-robin fashion. This may lead to a load imbalance: more data sets could be allocated to faster processors. But this would imply out-of-order execution and would require a complicated data management if, say, a replicated stage is followed by a non-replicated one in the application pipeline. In particular, large buffers would be required to ensure the in-order execution of the non-replicated stage.

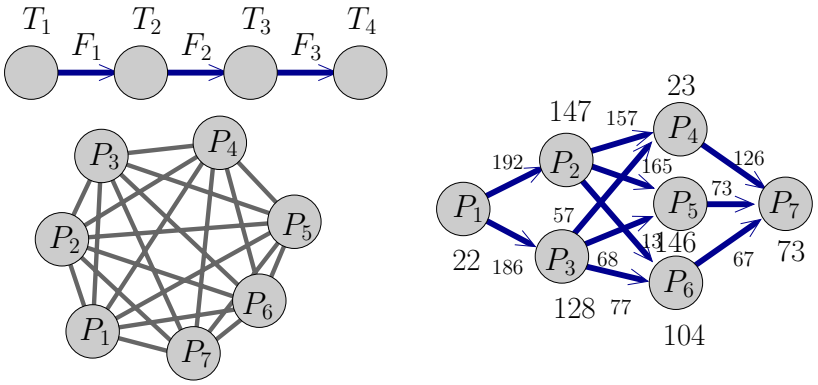

Figure 1: Example A: Four-stage pipeline, seven-processor computing platform, mapping with replication.

This explains why round-robin execution has been enforced in all the papers referenced above, and we enforce this rule too.

Because of the round-robin rule, the execution of a replicated stage is slowed down by the slowest processor involved in the round-robin. Let P_{slow} be the slowest processor involved in the replication of T_i. Then, if $p \in Team_i$, P_p processes one data set every R_i data sets at the speed dictated by P_{slow}, and thus its computation time (per data set) is $C_{\text{comp}}(p) = \frac{w_i}{R_i \times s_{slow}}$. Note that this implies that if processors of different speeds are processing a same stage, some of them will remain partly idle during the execution.

2.3 Throughput definition

The throughput ρ is defined as the average number of data sets which can be processed within one time unit. Equivalently, we aim at minimizing the period $\mathcal{P}$, which is the inverse of the throughput and corresponds to the time-interval that separates two consecutive data sets entering the system. We can derive a lower bound for the period as follows. Let $C_{\text{exec}}(p)$ be the cycle-time of processor P_p. If we enforce the **Overlap** model and constant communication and computation times, then $C_{\text{exec}}(p)$ is equal to the maximum of its reception time $C_{\text{in}}(p)$, its computation time $C_{\text{comp}}(p)$, and its transmission time $C_{\text{out}}(p)$: $C_{\text{exec}}(p) = \max\{C_{\text{in}}(p), C_{\text{comp}}(p), C_{\text{out}}(p)\}$. If we enforce the **Strict** model, then $C_{\text{exec}}(p)$ is equal to the sum of the three operations: $C_{\text{exec}}(p) = C_{\text{in}}(p) + C_{\text{comp}}(p) + C_{\text{out}}(p)$. Note that in both models, the maximum cycle-time, $\mathcal{M}_{\text{ct}} = \max_{1 \leq p \leq M} C_{\text{exec}}(p)$, is a lower bound for the period.

If no stage is replicated, then the throughput is simply determined by the critical resource (maximum cycle-time): $\rho = 1/\mathcal{M}_{\text{ct}}$. However, when stages are replicated, the previous result is no longer true, and more sophisticated techniques are required. In the following, we investigate how to compute the throughput when computation and communication times are constant or subject to random variations.

2.4 Random computation and communication times

We assume that at most one stage is processed by each processor. Since the mapping is given, the value of computation (or communication) times used by each resource fully defines the behavior of the system. Thus, we denote the computation time of stage T_i on processor P_p by $c_p = w_i/s_p$. Similarly, the communication time of the file F_i sent by P_p to P_q is given by $d_{p,q} = \delta_i/b_{p,q}$. We consider that the time to execute a stage, and the time to transfer data, are random variables.

Let $X_p(n)$ be the random variable giving the actual computation time of the n-th data set processed by P_p, where $p \in \mathit{Team}_i$ (recall that each processor deals with only one stage). In the deterministic case, we have $X_p(n) = w_i(n)/s_p(n)$ for all n, but in our probabilistic setting the $X_p(n)$ are random variables. Similarly, let $Y_{p,q}(n)$ be the random variable giving the actual communication time of the n-th file of type F_i transferred from P_p to P_q, where $p \in \mathit{Team}_i$ and $q \in \mathit{Team}_{i+1}$. In the deterministic case, $Y_{p,q}(n) = \delta_i(n)/b_{p,q}(n)$ for all n.

Two kinds of stochastic models are used in the following.

- The first one assumes that all the random sequences $\{X_p(n)\}_{n\in\mathbb{N}}$ and $\{Y_{p,q}(n)\}_{n\in\mathbb{N}}$ are I.I.D. (independent and identically distributed) and mutually independent sequences. This case is denoted as the *independent case*.
- The second case is more general: it assumes that the quantities defining the system, namely $\delta_i(n)$ and $w_i(n)$ as well as $s_p(n)$ and $b_{p,q}(n)$ are I.I.D. random processes, mutually independent, instead of the processing and communication times. This case is called the *associated case*, for reasons that will become clear in the following discussion.

Note that in both cases, we assume that the amount of work is independent from the amount of data to be communicated. This is the case for instance when data are compressed (and dependent of the data content), while the running time of the algorithm depends of the original data size.

The second case is more general than the first one since one can choose $\delta_i(n)$ and $w_i(n)$ to be constant (independent of n) so that $\{X_p(n)\}_{n\in\mathbb{N}}$ and $\{Y_{p,q}(n)\}_{n\in\mathbb{N}}$ become independent sequences. It should also be clear that this second case is more realistic. Imagine that the speed of the processors as well as the communication bandwiths on all the links are constant (this is typically the case when processors and resources are dedicated to this application and no other process or communication share the resources). Under such conditions, one should note that in the second case the variability on the processing and communication times only depend on the sizes and the bandwith requirements of the tasks. If one instance of task i happens to be large, say $\delta_i(n)$ is large, then, the processing time of task i on processor p, $X_p(n) = \delta_i(n)/s_p$ and the processing time of the same task on processor q at another stage of the pipeline, namely $X_q(n) = \delta_i(n)/s_q$ are related by $X_p(n) = \frac{s_q}{s_p} X_q(n)$ so that they are not independent. The I.I.D. assumption made in the first case only holds when the processing time of the same task over two different stages are not related in any manner.

The last remark, that will be developed later, concerns the relation between the sequences $\{X_q(n)\}_{n\in\mathbb{N}}$ and $\{X_p(n)\}_{n\in\mathbb{N}}$ in the second case. While the previous remark states that these two sequences are not independent, nothing is said

about the type of correlation that exists between them. Actually, it is rather direct to show that they are associated in the following informal sense: given two processors p and q, the larger $X_q(n)$ is, the larger $X_q(n)$ is likely to be, for all n. This will be given a formal meaning in Section 6.2.

We let (X, Y) denote the mapping of an application $(T_1, \ldots, T_N)$ on a platform $(P_1, \ldots, P_M)$. In both cases, since all variables are identically distributed, the probability that the computation time of T_i on P_p is larger than x does not depend on n and is given by $\Pr(X_p > x)$, while its expected value is denoted by $\mathbf{E}[X_p]$. So far, no assumption is made on the distributions of the random processing and communication times. However, some of our results are only valid for specific distributions of random variables. Below we recall the definition of these specific classes.

Exponential variables. An important class of random variables is the one of variables with exponential distribution. The probability that an exponential random variable X with a rate λ is larger than t is given by $\Pr(X > t) = e^{-\lambda t}$.

New Better than Used in Expectation variables. A random variable X is said to have a N.B.U.E. distribution if, and only if, $\mathbf{E}[X - t | X > t] \leq \mathbf{E}[X]$, for all $t > 0$. In other words, the N.B.U.E. assumption for communication or computation times means that if a computation (or a communication) has already been processed for a duration t and it is not finished yet, then the remaining time is smaller than the processing time of a fresh operation. This assumption is often true since in most cases, a partial execution of a stage should not increase the remaining work, especially when the amount of computation and communication are bounded from above. Note that exponential variables have the N.B.U.E. property, with equality in that case ($\mathbf{E}[X - t | X > t] = \mathbf{E}[X]$, for all $t > 0$). Also, note that there exist many statistical procedures to test if a random variable is N.B.U.E. ([17]).

In the following, we always assume, by default, that we are in the independent case, rather than in the associated case, unless it is stated otherwise.

3 Modeling the problem with event graphs

3.1 Mappings with replication

In this section, we aim at modeling mappings with timed Petri nets (TPNs) as defined in [1], in order to be able to compute the period of a given mapping. In the following only TPNs with the *event graph property* (each place has exactly one input and one output transition) will be considered (see [2]). We consider mappings where some stages may be replicated, as defined in Section 2.2: a stage can be processed by one or more processors. As already stated, two rules are enforced to simplify the model: a processor can process at most one stage, and if several processors are involved in the computation of one stage, they are served in a round-robin fashion. In all our Petri net models, the use of a physical resource during a time t (i.e., the computation of a stage or the transmission of a file from a processor to another one) is represented by a transition with a firing time t, and dependences are represented using places. Now, let us focus on the path followed in the pipeline by a single input data set, for a mapping with several stages replicated on different processors. Consider Example A described in Figure 1: the first data set enters the system and proceeds through processors

P_0, P_1, P_3 and P_6. The second data set is first processed by processor P_0, then by processor P_2 (even if P_1 is available), by processor P_4 and finally by processor P_6. There are 6 different paths followed by the data sets, and then data set i takes the same path as data set $i-6$. We have the following easy result:

Proposition 1. *Consider a pipeline of n stages T_1, ..., T_n, such that stage T_i is mapped onto m_i distinct processors. Then the number of paths followed by the input data in the whole system is equal to $m = \text{lcm}\,(m_1, \ldots, m_n)$.*

Proof. Let m be the number of paths $\mathcal{P}_j$ followed by the input data. Assume that stage T_i is processed by processors $P_{i,1}, \ldots, P_{i,m_i}$. By definition, all paths are distinct. Moreover, the round-robin order is respected: path $\mathcal{P}_j$ is made of processors $(P_{1,j \bmod m_1}, \ldots, P_{i,j \bmod m_i}, \ldots, P_{n,j \bmod m_n})$. The first path $\mathcal{P}_1$ is made of $(P_{1,1}, P_{2,1}, \ldots, P_{n,1})$. By definition, m is the smallest positive integer, such that the $(m+1)$-th used path is identical to the first one: $\forall i \in \{1, \ldots, n\}$, $m \bmod m_i = 0$. Indeed, m is the smallest positive integer, which is divisible by each m_i, i.e., $m = \text{lcm}\,(m_1, \ldots, m_n)$. □

The TPN model described below is the same flavor as what has been done to model jobshops with static schedules using TPNs [13]. Here, however, replication imposes that each path followed by the input data must be fully developed in the TPN: if P_1 appears in several distinct paths, as in Figure 1, there are several transitions corresponding to P_1. Furthermore, we have to add dependences between all the transitions corresponding to the same physical resource to avoid the simultaneous use of the same resource by different input data. These dependences differ according to the model used for communications and computations.

3.2 Overlap model

We first focus on the **Overlap** model: any processor can receive a file and send another one while computing. All paths followed by the input data in the whole system have to appear in the TPN. We use the notations of Proposition 1.

Let m denote the number of paths in the mapping. Then the i-th input data follows the $(i \bmod m)$-th path, and we have a rectangular TPN, with m rows of $2n-1$ transitions, due to the n transitions representing the use of processors and the $n-1$ transitions representing the use of communication links. The i-th transition of the j-th row is named T_i^j. The time required to fire a transition T_{2i}^j (corresponding to the processing of stage T_i on processor P_u) is set to $\frac{w_i}{s_u}$, and the one required by a transition T_{2i+1}^j (corresponding the transmission of file F_i from P_u to P_v) is set to $\frac{\delta_i}{b_{u,v}}$.

Then we add places between these transitions to model the following set of constraints:

1. The file F_i cannot be sent before the computation of T_i: a place is added from T_{2i}^j to T_{2i+1}^j on each row. Similarly, the stage T_{i+1} cannot be processed before the end of the communication of F_i: a place is added from T_{2i+1}^j to $T_{2(i+1)}^j$ on each row j.
2. When a processor appears in several rows, the round-robin distribution imposes dependences between these rows. Assume that processor P_i appears on rows $j_1, j_2, \ldots, j_k$. Then we add a place from $T_{2i}^{j_l}$ to $T_{2i}^{j_{l+1}}$ with $1 \leq l \leq k-1$, and a place from $T_{2i}^{j_k}$ to $T_{2i}^{j_1}$.

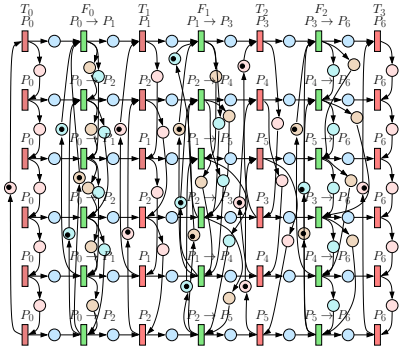

Figure 2: Complete TPN of Example A for the **Overlap** model.

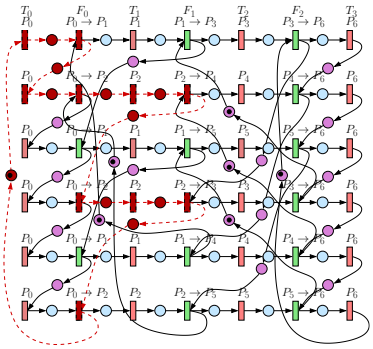

Figure 3: Complete TPN of Example A for the **Strict** model.

3. The one-port model and the round-robin distribution of communications also impose dependences between rows. Assume that processor P_i appears on rows $j_1, j_2, \ldots, j_k$. Then we add a place from $T_{2i+1}^{j_l}$ to $T_{2i+1}^{j_{l+1}}$ with $1 \leq l \leq k-1$, and a place from $T_{2i+1}^{j_k}$ to $T_{2i+1}^{j_1}$ to ensure that P_i does not send two files simultaneously, if P_i does not compute the last stage.
4. In the same way, we add a place from $T_{2i-1}^{j_l}$ to $T_{2i-1}^{j_{l+1}}$ with $1 \leq l \leq k-1$, and a place from $T_{2i-1}^{j_k}$ to $T_{2i-1}^{j_1}$ to ensure that P_i does not receive two files simultaneously, if P_i does not compute the first stage.

Finally, any resource before its first use is ready to compute or communicate, only waiting for the input file. Indeed, a token is put in every place going from a transition $T_i^{j_k}$ to a transition $T_i^{j_1}$, as defined in the previous lines. The complete TPN of Example A for the **Overlap** model is given in Figure 2.

3.3 Strict model

In the **Strict** model, any processor can either send a file, receive another one, or perform a computation while these operations were happening concurrently in the **Overlap** model. Hence, we require a processor to successively receive the data corresponding to an input file F_i, compute the stage T_{i+1} and send the

file F_{i+1} before receiving the next data set of F_i. Paths followed by the input data are obviously the same as in Subsection 3.2, and the structure of the TPN remains the same (m rows of $2n-1$ transitions).

The first set of constraints is also identical to that of the **Overlap** model, since we still have dependences between communications and computations, as in Figure 2. However, the other dependences are replaced by those imposed by the round-robin order of the **Strict** model. Indeed, when a processor appears in several rows, the round-robin order imposes dependences between these rows. Assume that processor P_i appears on rows $j_1, j_2, \ldots, j_k$. Then we add a place from $T_{2i+1}^{j_l}$ to $T_{2i-1}^{j_{l+1}}$ with $1 \leq l \leq k-1$, and a place from $T_{2i+1}^{j_k}$ to $T_{2i-1}^{j_1}$. These places ensure the respect of the model: the next reception cannot start before the completion of the current sequence reception-computation-sending. .

Any physical resource can immediately start its first communication, since it is initially only waiting for the input file. Thus a token is put in every place from a transition $T_i^{j_k}$ to a transition $T_i^{j_1}$, as defined in the previous lines. The complete TPN of Example A for the **Strict** model is given in Figure 3.

The automatic construction of the TPN in both cases has been implemented. The time needed to construct the Petri net is linear in its size: $\mathcal{O}(mn)$.

4 Computing throughputs, static case

TPNs with the event graph property make the computation of the throughput of a complex system possible through the computation of *critical cycles*, using $(\max, +)$ algebra [2]. For any cycle $\mathcal{C}$ in the TPN, let $\mathcal{L}(\mathcal{C})$ be its length (number of transitions) and $t(\mathcal{C})$ be the total number of tokens in places traversed by $\mathcal{C}$. Then a critical cycle achieves the largest ratio $\max_{\mathcal{C}\text{cycle}} \frac{\mathcal{L}(\mathcal{C})}{t(\mathcal{C})}$, and this ratio is the period $\mathcal{P}$ of the system: indeed, after a transitive period, every transition of the TPN is fired exactly once during a period of length $\mathcal{P}$ [2].

Critical cycles can be computed with softwares like ERS [14] or GreatSPN [8] with a complexity $\mathcal{O}(m^3n^3)$. By definition of the TPN, the firing of any transition of the last column corresponds to the completion of the last stage, i.e., to the completion of an instance of the workflow. Moreover, we know that all the m transitions (if m is still the number of rows of the TPN) of this last column are fired in a round-robin order. In our case, m data sets are completed during any period $\mathcal{P}$: the obtained throughput ρ is $\frac{m}{\mathcal{P}}$.

4.1 Overlap model

The TPN associated to the **Overlap** model has a regular structure, which facilitates the determination of critical cycles. In the complete TPN, places are linked to transitions either in the same row and oriented forward, or in the same column. Hence, any cycle only contains transitions belonging the same "column": we can split the complete TPN into $2n-1$ smaller TPNs, each sub-TPN representing either a communication or a computation. However, the size of each sub-TPN (the restriction of the TPN to a single column) is not necessarily polynomial in the size of the instance, due to the possibly large number of rows, equal to $m = \text{lcm}\,(m_0, \ldots, m_{n-1})$.

It turns out that a polynomial algorithm exists to find the weight $\mathcal{L}(\mathcal{C})/t(\mathcal{C})$ of a critical cycle: only a fraction of each sub-TPN is required to compute this weight, without computing the cycle itself.

Theorem 1. *Consider a pipeline of n stages $T_1, \ldots, T_n$, such that stage T_i is mapped onto m_i distinct processors. Then the average throughput of this system can be computed in time $\mathcal{O}\left(\sum_{i=1}^{n-1}\left((m_i m_{i+1})^3\right)\right)$.*

Due to lack of space, the proof is available in the Web supplementary material, or in [4].

In Example A (see Figure 1), a critical resource is the output port of P_1, whose cycle-time is equal to the period, 189. However it is possible to exhibit cases without critical resource.

4.2 Strict model

Cycles in the TPN associated to the **Strict** model are more complex and less regular, since corresponding TPNs have backward edges. The intuition behind these backward edges is that a processor P_u cannot compute an instance of S_i before having completely sent the result F_i of the previous instance of S_i to the next processor P_v. Thus, P_u can be slowed by P_v. As for the **Overlap** model, there exist mappings for which all resources have idle times during a complete period. With the **Strict** model, this is the case for Example A, the critical resource is P_2, which has a cycle-time $\mathcal{M}_{\text{ct}} = 215.8$, strictly smaller than the period $\mathcal{P} = 230.7$.

5 Computing the throughput with exponential times

In this section, we consider the case of exponential laws: all processing times and communication times are exponentially distributed. In the corresponding Petri net, all transitions (modeling processing times or modeling communication times) have exponential firing times. The probability of the firing time t_i of a transition is given by $\Pr(t_i > x) = 1 - e^{-\lambda_i x}$. The firing rate λ_i corresponds either to the processing rate of one processor or the communication rate over one link.

The study of the exponential case is motivated by two facts. First, one can get explicit formulas in this case. Second (as we will see in Section 6), exponential laws are extreme cases among all N.B.U.E. variables.

In the rest of this section, we first explain the general method which allows us to compute the throughput for both the **Overlap** and the **Strict** models. However, this general method has a high complexity. In the **Overlap** case, we provide a simpler method, building upon the relative simplicity of the timed Petri net (Section 5.2). Finally in Section 5.3, we derive a polynomial algorithm for the **Overlap** case when we further assume that the communication network is homogeneous.

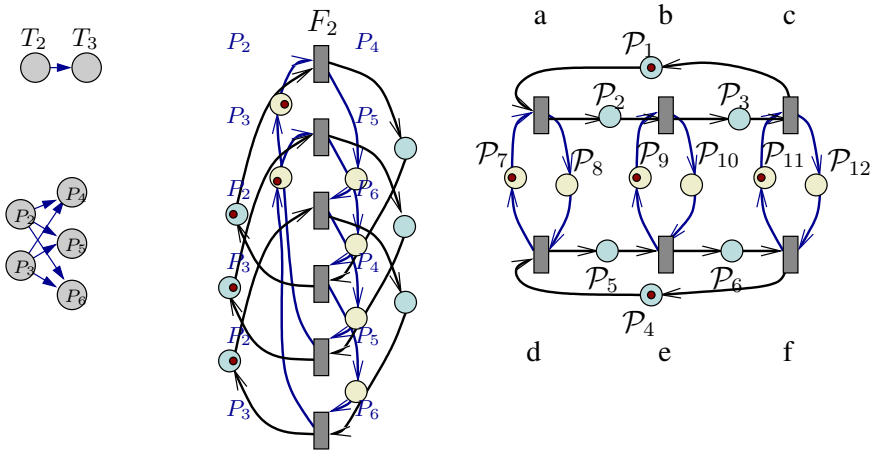

Figure 4: Example A: Part of the timed Petri net corresponding to communication F_2.

5.1 General method to compute the throughput

Theorem 2. *Let us consider the system (X, Y) formed by the mapping of an application onto a platform. Then the throughput can be computed in time $O\left(\exp(\mathrm{lcm}_{1\leq i\leq N}(R_i))^3\right)$.*

Proof. The whole proof is made of four successive steps.

Model the system as a timed Petri net The transformation of the initial system into a timed Petri net is fully described in Section 3. Recall from Section 3 that it consists in $R = \mathrm{lcm}_{1\leq i\leq N}(R_i)$ rows and $2N-1$ columns, and examples for both models are depicted in Figure 2 and Figure 3. This step is done in time $O(RN)$, and the expectation of the delay between two successive firings of any transition gives the throughput of the system.

Transformation of the timed Petri net into a Markov chain To compute the expectation of the delay between two successive firings of any transition, we transform the above timed Petri net into a Markov chain $(Z_1, Z_2, \ldots)$. To each possible marking of the timed Petri net, we associate a state x_i. There are $(2N + 3(N-1))R$ places, and each place contains either zero or one token. Thus, there are at most $2^{(2N+3(N-1))R}$ possible different markings, leading to the same number of states in the Markov chain.

Due to the exponential size of the number of states of the Markov chain, we only consider the part of the timed Petri net corresponding to communications in examples. This part is shown in Figure 4.

On Example A, places are named ($\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$, $\mathcal{P}_4$, $\mathcal{P}_5$, $\mathcal{P}_6$, $\mathcal{P}_7$, $\mathcal{P}_8$, $\mathcal{P}_9$, $\mathcal{P}_{10}$, $\mathcal{P}_{11}$, $\mathcal{P}_{12}$), while transitions are named (a, b, c, d, e, f). Thus, a state is defined by a 12-uple, each number equal to either 0 or 1 being the number of tokens in the place. In the Markov chain, moving from a state to another corresponds to the firing of a transition of the timed Petri net. Thus, arrows in the graphical

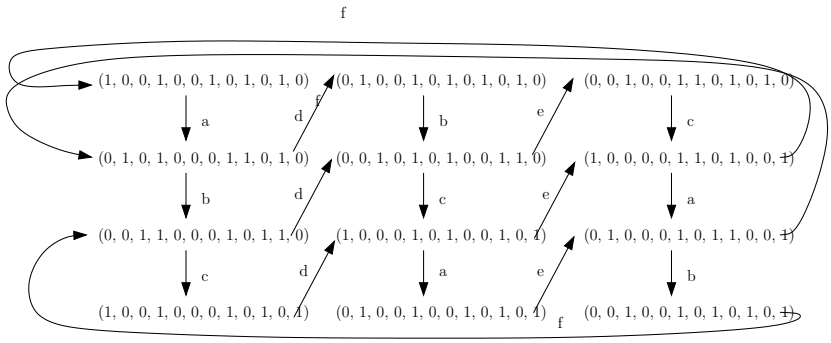

Figure 5: List of all possible states of the Markov chain corresponding to the reduced timed Petri net of Example A.

representation of the Markov chain are labeled with the names of the transitions. The complete list of possible states and the corresponding transitions are given in Figure 5.

If in state x_i, transition $\mathcal{T}_j$ can be fired leading to state x_k, then the transition rate of the corresponding arrow is set to λ_j.

Computation of the throughput Using this new representation, we are able to compute the throughput. The throughput is the number of completed last stage T_N per time unit. In terms of Petri nets, this is also the expected number of firings per time unit of the transitions in the last column. Thus, in terms of Markov chains, the throughput is given by the probability of being in one of the states enabling these transitions. By construction of the Markov chain, all of its states are positive recurrent. Thus, it admits a stationary distribution, giving the probability of each state. This stationary distribution can be computed in polynomial time in the size of the Markov chain by solving a linear system [12]. The sum of the probability of the valid states returns the throughput. □

5.2 Overlap model

We now focus on the **Overlap** model. As in the deterministic case, constraints applying to our system form a very regular timed Petri net which is feed forward (dependencies only from column C_i to column C_{i+1}, for $1 \leq i \leq 2N-2$), giving an easier problem than the **Strict** model.

Theorem 3. *Let us consider the system (X, Y) formed by the mapping of an application onto a platform, following the **Overlap** communication model. Then the throughput can be computed in time $O\left(N \exp(\max_{1 \leq i \leq N}(R_i))\right)$.*

Proof. First, let us give the overall structure of the proof:

1. split the timed Petri net into columns C_i, with $1 \leq i \leq 2N-1$;
2. separately consider each column C_i;
3. separately consider each connected component D_j of C_i;
4. note that each component D_j is made of many copies of the same pattern $\mathcal{P}_j$, of size $u_j \times v_j$;

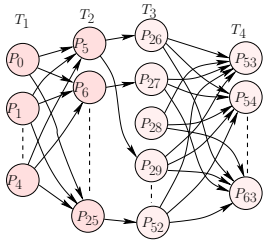

Figure 6: Example C: Stages are respectively replicated on 5, 21, 27 and 11 processors.

5. transform $\mathcal{P}_j$ into a Markov chain $\mathcal{M}_j$;
6. determine a stationary measure of $\mathcal{M}_j$, using a combinatorial trick based on Young diagrams [16];
7. compute the throughput of $\mathcal{P}_j$ in isolation (called hereinafter inner throughput of component D_j);
8. combine the inner throughputs of all components to get the global throughput of the system.

To decrease the overall complexity, we use the same idea as in Section 4.1: thanks to the regularity of the global timed Petri net, we split it into a polynomial number of columns, and we compute the throughput of each column independently.

Let us focus on a single column. We have two cases to consider: (i) the column corresponds to the computation of a single processor (columns C_{2i-1}, for $1 \leq i \leq N$); (ii) the column corresponds to communications between two sets of processors (columns C_{2i}, for $1 \leq i \leq N-1$).

In case (i), cycles do not interfere: any cycle involves a single processor, and any processor belongs to exactly one cycle. Thus, the inner throughput is easily computed, this is the expectation of the number of firings per time unit. The processing time $X_p(n)$ being exponential, this is equal to the rate λ_p of X_p.

On the contrary, case (ii) is more complex and requires a more detailed study. Let us consider the i-th communication (it corresponds to column C_{2i}): it involves R_i senders and R_{i+1} receivers. We know from its structure that the timed Petri net is made of $g = \gcd(R_i, R_{i+1})$ connected components. Let u_j be equal to R_i/g and v_j be equal to R_{i+1}/g. Then each connected component D_j in this column is made of $c = \frac{R}{\mathrm{lcm}(R_i, R_{i+1})}$ copies of a pattern $\mathcal{P}_j$ of size $u_j \times v_j$. Since these components are independent, we can compute the throughput of each of them independently. In the case of Example C presented in Figure 6, we consider a 4-stage application, such that stages are replicated on respectively 5, 21, 27 and 11 processors. More precisely, we focus on the second communication (see Figure 7, involving 21 senders and 27 receivers). In this case, we have $g = 3$ connected components, made of 55 copies of pattern $\mathcal{P}_j$ of size $u_j \times v_j = 9 \times 7$.

Each pattern is a timed Petri net $\mathcal{P}_j$ with a very regular structure, which can be represented as a rectangle of size (u_j, v_j), also denoted (u, v) to ease notations, as shown in Figure 7. As said before, determining the throughput of $\mathcal{P}$ is equivalent to determining a stationary measure of a Markov chain. We know that the stationary measure of a Markov chain with t states can be computed

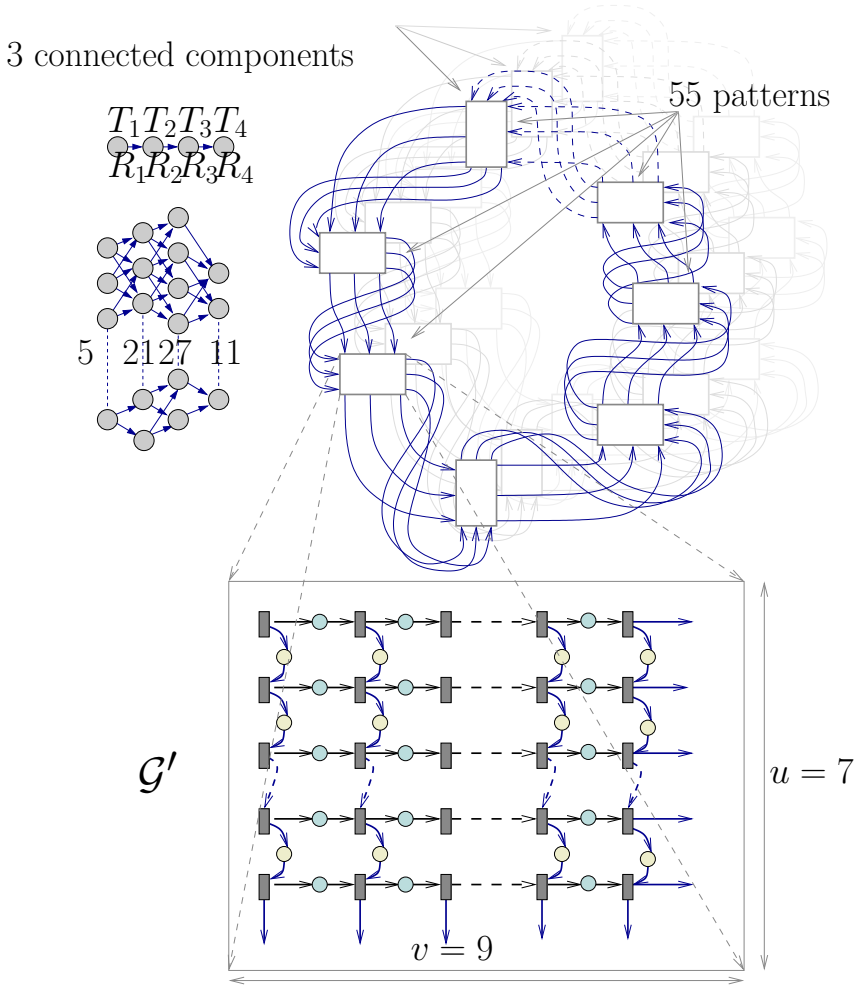

Figure 7: Example C, with stages replicated on 5, 21, 27 and 11 processors, and structure of the timed Petri net corresponding to the second communication.

in time $O\left(t^3\right)$ [12]. Thus, we need to determine the number of states of the transformation of $\mathcal{P}_j$ into a Markov chain. Let $\mathcal{M}_j$ be this Markov chain.

The number of states of $\mathcal{M}_j$ is by definition the number of possible markings, and we can directly determine it. A valid marking of $\mathcal{P}_j$ is represented in Figure 8. The regularity of the structure imposes some constraints to valid markings: a transition can be fired for the k-th time if, and only if, all the transitions above it or on its left have been fired k times. In other terms, if a processor sends a file to q receivers $P_1, \ldots, P_q$, it can send the k-th instance of the file to P_i if and only if it has sent the k first instances of the file to $P_1, \ldots, P_{i-1}$.

In our rectangular representation of the timed Petri net, the borderline between transitions that have been fired $k+1$ times and those that have been fired k times is the union of two Young diagrams, as displayed on Figure 8. Since there is only a single token in each column and in each row, we cannot have three simultaneous Young diagrams.

Let us compute the number of states of the Markov chain $\mathcal{M}_j$. As said in the previous paragraph, the borderline can be seen as two Young diagrams, or two paths. The first one is from coordinates $(i, 0)$ to $(0, j)$, and the second one goes from (u, j) to (i, v) (see Figure 9). If i and j are given, then there are $\alpha_{i,j} = \left(\begin{array}{c} i+j \\ i \end{array} \right)$ possible paths from $(i, 0)$ to $(0, j)$, where $\left(\begin{array}{c} n \\ k \end{array} \right)$ is equal to $\frac{n!}{k!(n-k)!}$. Similarly, there are $\alpha_{u-1-i,v-1-j}$ possible paths from (u, j) to (i, v). Thus, if i and j are given, then there are $\alpha_{i,j} \times \alpha_{u-1-i,v-1-j}$ possible markings. If i and j are not given anymore, then the total number $S(u, v)$ of valid markings can be easily determined:

$$\begin{aligned} S(u,v) &= \sum_{i=0}^{u-1} \sum_{j=0}^{v-1} \alpha_{i,j} \alpha_{u-1-i,v-1-j} \\ &= \sum_{i=0}^{u-1} \sum_{j=0}^{v-1} \left(\begin{array}{c} i+j \\ i \end{array} \right) \left(\begin{array}{c} u+v-2-i-j \\ u-1-i \end{array} \right) \\ &= \left(\begin{array}{c} u+v-1 \\ u-1 \end{array} \right) v = \frac{(u+v-1)!}{(u-1)!v!} v \ . \end{aligned}$$

Thus, the final Markov chain of a single connected component has exactly $S(u, v) = \frac{(u+v-1)!}{(u-1)!v!} v$ states, and its inner throughput can be computed in time $S(u, v)^3$.

Let us now come to the computation of the global throughput of the system. Actually, the throughput is given by the following iteration. The throughput of one strongly connected component is the minimum of its inner throughput and the throughput of all its input components, so once all inner throughputs are known, the computation of the throughput is linear in the number of components.

In column C_{2i}, we have $g = \gcd(R_i, R_{i+1})$ connected components so that the total computation time to obtain their throughput is equal to $gS(u, v)$. Since we have $S(gu, gv) \geq gS(u, v)$, $u = R_i/g$ and $v = R_{i+1}/g$, the total computation time to determine the throughput of C_{2i} is less than $S(R_i, R_{i+1})$.

Finally, the total computation time of the throughput is equal to $\sum_{i=1}^{N-1} S(R_i, R_{i+1})^3$, leading to our result of a throughput that can be computed in time $O\left(N \exp(\max_{1 \leq i \leq N}(R_i))\right)$.

□

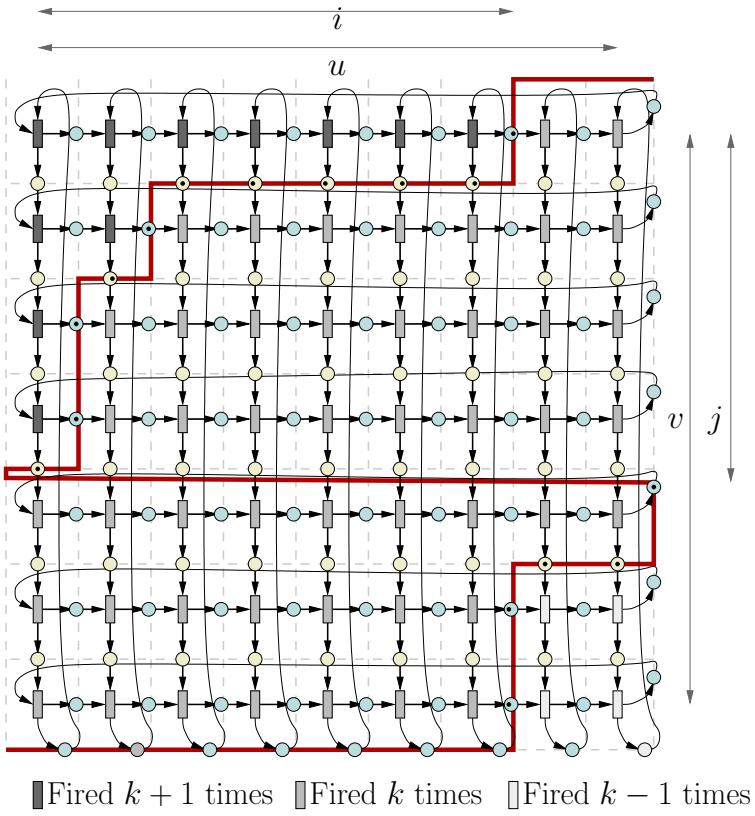

Figure 8: Valid marking of $\mathcal{P}_j$, the reduced timed Petri net of the second communication of Example C.

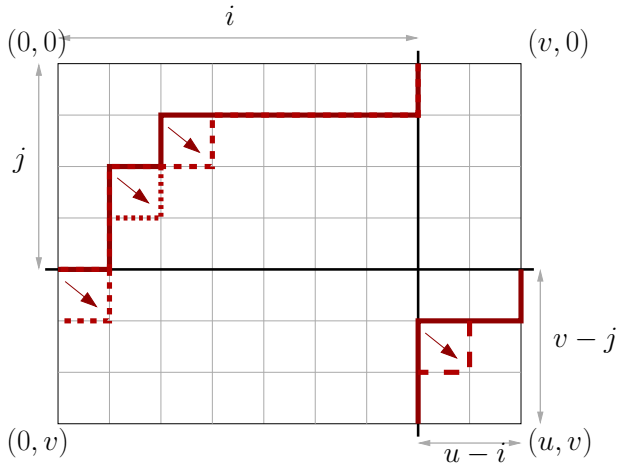

Figure 9: Reachable states from a given position.

5.3 Overlap model, homogeneous communication network

In the case where all the communication times in one column are all I.I.D., with the same rate in component D_j, denoted λ_j, then the inner throughput of each strongly connected component (i.e., the throughput of the component if isolated from the global timed Petri net) can be computed explicitly with a very simple formula. This reduces the overall computation of the throughput to a simple computation of minimums over the strongly connected components, which can be done in polynomial time.

Theorem 4. *Let us consider the system (X, Y) formed by the mapping of an application onto a platform, following the **Overlap** communication model with a homogeneous communication network. Then the throughput can be computed in polynomial time.*

The global throughput can then be computed in polynomial time from all inner throughputs, and it is equal to $\rho = \sum_{D_j \in C_{2N-1}} \min_{D_{j'} \prec D_j} \rho_{j'}$, where $D_{j'} \prec D_j$ means that there exists a path from component $D_{j'}$ to component D_j, or $D_{j'} = D_j$. Because of the structure of the timed Petri net, if $D_{j'}$ is in column $C_{i'}$ and D_j is in column C_i, then $i' < i$ or $j' = j$. The computation can thus be done column by column. For any component in the first column, its throughput must be equal to its inner throughput ρ_j. The computation for column i only depends on results from column $i-1$ by construction of the Petri net. Moreover, the total number of components is polynomial in the number of processors M. We obtain therefore a polynomial complexity ($2N-1$ columns in the timed Petri net, and a polynomial number of components).

Proof. Platforms with the **Overlap** model and a homogeneous communication network are special cases of the **Overlap** model. Thus, the demonstration of Theorem 3 remains true, and we focus again on the Markov chain $\mathcal{M}_j$, obtained from a pattern of component D_j.

If D_j corresponds to a processor, the formula given previously applies, and its inner throughput is $\rho_j = \lambda_j$.

Next, we focus on a strongly connected component corresponding to a communication. We already know that the throughput is given by an invariant measure of $\mathcal{M}_j$. Graphically, the set of reachable states from a given state is easy to define: any of the top-left corners in the line can be "inverted" into a bottom-right corner to obtain a new valid state. In terms of Petri nets, this corresponds to the firing of one fireable transition. On Figure 8, there are 4 fireable transitions, corresponding to 4 top-left corners on the bold line in the equivalent Figure 9. Moreover, this reasoning can be inverted: any bottom-right corner in Figure 8 can be inverted, giving a possible previous marking leading to the current marking. Since we have as many top-left corners as bottom-right ones on Young diagrams, any state of $\mathcal{M}_j$ has the same number of incoming states as outgoing ones. Moreover, since the communication network is homogeneous, all transitions have the same firing rate. These two conditions imply that the invariant measure of the Markov chain $\mathcal{M}_j$ is uniform [12]: if S is the number of states of $\mathcal{M}_j$, then its invariant measure is $(\frac{1}{S}, \ldots, \frac{1}{S})$.

Last, let us compute the number of states of $\mathcal{M}_j$ allowing a given transition to be fired. Due to symmetry, all transitions have the same firing rate and we can only consider the top-right transition $\mathcal{T}_0$ of the net. By using the bijection with Young diagrams, the number of markings such that $\mathcal{T}_0$ is fireable is

exactly the number $S'(u,v)$ of possible paths starting from this top-right corner. The quantity $S'(u,v)$ is computed in the same way as $S(u,v)$ (see proof of Theorem 3):

$$S'(u,v) = \sum_{i=0}^{u-2}\sum_{j=0}^{v-2}\alpha_{i,j} = \left(\begin{array}{c} u+v-2 \\ u-1 \end{array}\right) = \frac{S(u,v)}{v+u-1}\,.$$

Finally, we know the rate of the states leading to a given transition, and the number of states leading to it. Between two successive firings of the same transitions, uv communications are made (remind that $\gcd(u,v)=1$). Thus, the inner throughput is equal to $\frac{vu\lambda_j}{(v+u-1)}$.

As in the previous case, the throughput of a component can be computed in an iterative way. The throughput of one component is equal to the minimum of its inner throughput and the throughput of all its incoming components. This allows one to compute the throughput of all components starting from the first column and ending in the last one.

Now, the global throughout is the rate at which tasks exit the system. This is equal to the sum of the throughputs of all components in the last column $2N-1$. The throughputs of the last components are equal to the minimum of the inner throughputs of all components on paths from the first column to the last.

Computing ρ column by column renders the computation of this formula polynomial in the number of tasks and processors. □

6 Comparison results in case of general I.I.D. variables

In the previous section, we have shown how to compute the throughput when all communication times and all processing times are exponential variables (and this even in polynomial time for the homogeneous **Overlap** case). In general, it is well known that the computation of the throughput is hard for arbitrary random communication times and processing times, even for very simple cases [15]. However, the fact that in our case, the throughput is an increasing and convex function of communication times and processing times implies that one can use stochastic comparisons to construct bounds on the throughput in the case where communication times and processing times are I.I.D. N.B.U.E. variables (see Section 6). Moreover, the lower and upper bounds are obtained by the deterministic and exponential cases respectively.

This section is dedicated to the study of several comparisons between systems with different processing and communication times. To do so, we first need to introduce a way to define orders on random sequences. We use classical definitions.

Definition 1. *Let $\{V(n)\}_{n\in\mathbb{N}}$ and $\{W(n)\}_{n\in\mathbb{N}}$ be two real random variable sequences:*
• *V is smaller than W for the strong order (denoted $V \leq_{\mathrm{st}} W$) if for all increasing function f,*
$\mathbf{E}[f(V(1),V(2),\cdots)] \leq \mathbf{E}[f(W(1),W(2),\cdots)]$.
• *V is smaller than W for the increasing convex order (hereinafter denoted*

$V \leq_{\text{icx}} W$*) if for all increasing convex function* g*, we have* $\mathbf{E}[g(V(1), V(2), \cdots)] \leq \mathbf{E}[g(W(1), W(2), \cdots)]$.

We also consider another order for multi-dimensional variables which is weaker than the strong order but that will be enough to compare the expected throughputs.

Definition 2. *Let* $\{V(n)\}_{n\in\mathbb{N}}$ *and* $\{W(n)\}_{n\in\mathbb{N}}$ *be two random variable sequences in* $\mathbb{R}^k$*:*
V *is smaller than* W *for the lower orthant order (denoted* $V \leq_{lo} W$*) if for any* n*,*

$$P(V(1) \leq x_1, V(2) \leq x_2, \cdots) \geq P(W(1) \leq x_1, W(2) \leq x_2, \cdots).$$

6.1 The I.D.D. case

In the following, we consider a very general system that is either **Strict** or **Overlap** and whose processing times and communication times are I.I.D..

Theorem 5. *Consider two systems* $(X^{(1)}, Y^{(1)})$ *and* $(X^{(2)}, Y^{(2)})$*. If we have for all* n*,* $\forall 1 \leq p \leq M$, $X_p^{(1)}(n) \leq_{\text{st}} X_p^{(2)}(n)$ *and* $\forall 1 \leq p, q \leq M, Y_{p,q}^{(1)}(n) \leq_{\text{st}} Y_{p,q}^{(2)}(n)$*, then* $\rho^{(1)} \geq \rho^{(2)}$.

Proof. Consider the Petri nets modeling both systems. They only differ by the firing times of the transitions. Then for $b = 1, 2$, let $D_k^b(n)$ be the time when transition $\mathcal{T}_k$ ends its n-th firing. The Petri net being an event graph (all places have a single input transition and all places have a single output transition), the variables $D_k^b(n)$ satisfy a (max,plus) linear equation[1]: $D^b(n) = D^b(n-1) \otimes A^b(n)$, where the matrices $A^b(n)$ are such that $A^b(n)_{ij} = \sum_k T_k^b(n)$ if a path connects transtions $\mathcal{T}_p$ and $\mathcal{T}_q$ with one token in the first place of the path and no token in any other place. Now, the firing times of the transitions $T_k^b(n)$ are either communication times or processing times so that there exists i, j (only depending on k, in a bijective way) such that $T_k^b(n) = X_p^{(b)}(n)$ or $T_k^b(n) = Y_{p,q}^{(b)}(n)$. Therefore, $T_k^1(n)$ and $T_k^2(n)$ are I.I.D. sequences such that $T_k^1(n) \leq_{st} T_k^2(n)$ for all n and k, so that the same holds for the sequence of matrices $A^b(n)$. Now, the (max,plus) matrix product and the sum are increasing functions. This implies that $D^1(n) \leq_{\text{st}} D^2(n)$.

Finally, the throughput $\rho^{(b)}$ is the limit of $n/\mathbf{E}[D^b(n)]$ when n goes to infinity, so that $\rho^{(1)} \geq \rho^{(2)}$, which concludes the proof. □

Theorem 6. *Let us consider two systems with I.I.D. communication and processing times* $(X^{(1)}, Y^{(1)})$ *and* $(X^{(2)}, Y^{(2)})$*. If we have for all* n*,* $\forall 1 \leq p \leq M, X_p^{(1)}(n) \leq_{\text{icx}} X_p^{(2)}(n)$ *and* $\forall 1 \leq p, q \leq M$, $Y_{p,q}^{(1)}(n) \leq_{\text{icx}} Y_{p,q}^{(2)}(n)$*, then* $\rho^{(1)} \geq \rho^{(2)}$.

Proof. The proof is similar to the previous one, using the fact that $D_k^b(n)$ is also a convex function (a composition of maximum and sums) of the communication and processing times. □

[1] The product $\otimes$ is defined as: $(V \otimes M)_k = \max_i(V_i + M_{ik})$.

Theorem 7. *Let us consider any system* $(X^{(1)}, Y^{(1)})$*, such that* $X_p^{(1)}(n)$ *and* $Y_{p,q}^{(1)}(n)$ *are N.B.U.E.. Let us also consider two new systems* $(X^{(2)}, Y^{(2)})$ *and* $(X^{(3)}, Y^{(3)})$ *such that:*

- $\forall 1 \leq p \leq M, X_p^{(2)}(n)$ *has an exponential distribution, and* $\mathbf{E}[X_p^{(2)}(n)] = \mathbf{E}[X_p^{(1)}(n)]$,
- $\forall 1 \leq p, q \leq M, Y_{p,q}^{(2)}(n)$ *has an exponential distribution, and* $\mathbf{E}[Y_{p,q}^{(2)}(n)] = \mathbf{E}[Y_{p,q}^{(1)}(n)]$,
- $\forall 1 \leq p \leq M, X_p^{(3)}(n)$ *is deterministic and for all integers* n, $X_p^{(3)}(n) = \mathbf{E}[X_p^{(1)}(n)]$,
- $\forall 1 \leq p, q \leq M, Y_{p,q}^{(3)}(n)$ *is deterministic and for all* n, $Y_{p,q}^{(3)}(n) = \mathbf{E}[Y_{p,q}^{(1)}(n)]$.

Then we have $\rho^{(3)} \geq \rho^{(1)} \geq \rho^{(2)}$.

Proof. A direct consequence of the N.B.U.E. assumption is that if V is N.B.U.E. and W is exponential with the same mean as V, then $V \leq_{icx} W$ (see [11], for example). It is also direct to show that if U is deterministic and $U = \mathbf{E}[V]$, then $U \leq_{icx} V$. Therefore, a direct application of Theorem 6 shows that $\rho^{(3)} \geq \rho^{(1)} \geq \rho^{(2)}$. □

In particular, Theorem 7 implies that in the **Overlap** case with a homogeneous communication network, as soon as communication times and processing times are N.B.U.E., then the throughput ρ can be bounded explicitly. It is comprised between the throughput of the system in which all random processing times are replaced by their mean values, where the inner throughput of processing components are the same as in the exponential case and the throughput of communication components is replaced by $\frac{u_i v_i \lambda_i}{\max(u_i, v_i)} = \min(u_i, v_i)\lambda_i)$, and the throughput of the system in which all random processing times are replaced by exponential variables with the same mean value.

6.2 The associated case

In this section, we study the case where the computation and the communication times are not mutually independent but associated.

The first following lemma establishes the formal association properties between the processing times (resp. communication times) on different processors (resp. links).

Lemma 1. *For any processors* p *and* q*, the processing times* $X_q(n)$ *and* $X_q(n)$ *satisfy*

$$P(X_q(n) < t, X_q(n) < s) \geq P(X_p(n) < t)P(X_q(n) < s).$$

The same holds for the communication times on two different links (p, q) *and* (p', q')*:*

$$P(Y_{p,q}(n) < t, Y_{p',q'}(n) < s) \geq P(X_p q(n) < t)P(X_{p'} q'(n) < s).$$

Proof. The proof uses the definition of the variables. It is carried for the computation times only since the proof for the communication times is exactly the same. First, if p and q do not process the same task (*i.e.* $i(p) \neq i(q)$), then $X_p(n)$ and $X_q(n)$ are independent and therefore satisfy the inequality as an equality. If $i(q) = i(p)$ (denoted i in the rest of the proof), Then, by definition,

$X_p(n) = \delta_i(n)/s_p(n)$ and $X_q(n) = \delta_i(n)/s_q(n)$. Therefore, for the association property,

$$\begin{aligned}
&\Pr\left(X_p(n) < t, X_q(n) < s\right)\\
&= \Pr\left(\delta_i(n)/s_p(n) < t, \delta_i(n)/s_q(n) < s\right),\\
&= \Pr\left(\delta_i(n) < \min(ts_p(n), ss_q(n))\right),\\
&\geq \Pr\left(\delta_i(n) < ts_p(n)\right)\Pr\left(()\,\delta_i(n) < ts_q(n)\right),
\end{aligned}$$

by independence of $s_p(n)$ and $s_q(n)$. □

Let us denote by $\{X_p(n)^*_{1\leq p\leq M, n\in\mathbb{N}}\}$ (resp. $\{Y_{q,p}(n)^*_{1\leq p,q\leq M,n\in\mathbb{N}}\}$) the set of processing times (resp. comunication times) with the same distribution as $\{X_p(n)^*_{1\leq p\leq M,n\in\mathbb{N}}\}$ (resp. $\{Y_{q,p}(n)^*_{1\leq p,q\leq M,n\in\mathbb{N}}\}$), but mutually independent.

Lemma 2. *Under the foregoing notations,*

$$\begin{aligned}
\{X_p(n)_{1\leq p\leq M,n\in\mathbb{N}}\} &\leq_{lo} \{X_p(n)^*_{1\leq p\leq M,n\in\mathbb{N}}\},\\
\{Y_{q,p}(n)_{1\leq p,q\leq M,n\in\mathbb{N}}\} &\leq_{lo} \{Y_{q,p}(n)^*_{1\leq p,q\leq M,n\in\mathbb{N}}\}.
\end{aligned}$$

Proof. The proof comes from a direct generalisation of Lemma 1 from two processors to M processors. □

Theorem 8. *Let us consider any system $(X^{(1)}, Y^{(1)})$, such that $X_p^{(1)}(n)$ and $Y_{p,q}^{(1)}(n)$ are associated . Let us also consider two new systems $(X^{(2)}, Y^{(2)})$ and $(X^{(3)}, Y^{(3)})$ such that:*

- *$X_p^{(2)}(n)$ and $Y_{p,q}^{(2)}(n)$ are iid with the same distribution as in $(X^{(1)}, Y^{(1)})$.*
- *$\forall 1 \leq p, q \leq M, Y_{p,q}^{(3)}(n)$ is deterministic and for all n, $Y_{p,q}^{(3)}(n)$ and $X_p^{(3)}(n)$ are deterministic and for all integers n, $X_p^{(3)}(n) = \mathbf{E}[X_p^{(1)}(n)]$, and $Y_{p,q}^{(1)}(n) = \mathbf{E}[Y_{p,q}^{(1)}(n)]$.*

Then we have $\rho^{(3)} \geq \rho^{(1)} \geq \rho^{(2)}$.

Proof. The proof follows the same ideas as the proof of Theorem 5. The first inequality comes from Jensen inequality that also holds for associated variables. Since the throughput is the limit of an increasing convex functions of the variables $Y_{p,q}^{(1)}(n)$ and $X_p^{(1)}(n)$, then its expectation satisfies the first inequality. As for the second inequality, it uses directly Lemma 10.3 from [11], to get a comparison for the firing times of all the transitions in the Petri net model for both systems: $D_k^1(n) \leq_{lo} D_k^2(n)$. The throughput being the limit of $n/\mathbf{E}[D^b(n)]$ when n goes to infinity, one gets directly $\rho^{(1)} \geq \rho^{(2)}$. □

When the task sizes (resp. bandwith requirements) have the N.B.U.E. property while the processor speeds and the communication speeds can be arbitrary (but uniformly bounded away from 0, by, say, b), with a density f_{s_p}, then the processing times as well as the communication times become DFR, which implies the N.B.U.E. property.

Indeed, by definition of N.B.U.E. variables, $\int_s \Pr\left(\delta_i > t + s | \delta_i > t\right) ds \leq \int_s \Pr\left(\delta_i > s\right) ds$.

Let us consider the processing time of one task (the case of communication times is similar): $X_p = \delta_{i(p)}/s_p$ and let us compute (by setting $i = i(p)$),

$$\begin{aligned}
&\int_{s=0}^{\infty} \Pr\left(X_p > t+s | X_p > t\right) ds \\
&= \int_{s=0}^{\infty} \Pr\left(\delta_i/s_p > t+s | \delta_i/s_p > t\right) ds \\
&= \int_{s=0}^{\infty} \Pr\left(\delta_i > s_p(t+s) | \delta_i > s_p t\right) ds \\
&= \int_{s=0}^{\infty} \int_{u=b}^{\infty} \Pr\left(\delta_i > ut+us | \delta_i > ut, s_p = u\right) f_{s_p}(u) du ds \\
&= \int_{s=0}^{\infty} \int_{u=b}^{\infty} \frac{\Pr\left(\delta_i > ut+us | s_p = u\right)}{\Pr\left(\delta_i > ut | s_p = u\right)} f_{s_p}(u) du ds \\
&= \int_{u=b}^{\infty} f_{s_p}(u) \int_{s=0}^{\infty} \frac{\Pr\left(\delta_i > u(t+s)\right)}{\Pr\left(\delta_i > ut\right)} ds du \\
&\leq \int_{u=b}^{\infty} f_{s_p}(u) \int_{s=0}^{\infty} \Pr\left(\delta_i > us\right) ds du \\
&= \int_{s=0}^{\infty} \Pr\left(X_p > s\right) ds.
\end{aligned}$$

By combining Theorem 8 and Theorem 7, one gets the following result, which the final point of this section.

If all processors are independent of each other and the tasks are independent and have sizes with the N.B.U.E. property, then, the throughput of the system is bounded from below by a version of the system where the processing times and the communication times are replaced by I.I.D. exponential variables with the same means and is bounded from above by another version of the system where all processing times and communication times are replaced by constants (equal to the mean).

This justifies the focus on the constant and exponential cases, as extreme cases.

7 Experimental results

To assess the quality of our model and to observe the actual behaviour of several distributions, we made a lot of simulations, using very different ways. The event graph model presented in Section 3 is simulated using the toolbox ERS [14]. Two tools were used, the first one, scscyc, is able to determine the cycle time in the deterministic case, while the second one, eg_sim, determines the cycle time in both deterministic and exponential cases. The throughput of this second method relies on the number of simulated events. The whole system is simulated using the Simgrid framework [7]. Since Simgrid aims at being a realistic simulation, any communication cannot use more than 92% of the theoretical bandwidth [22]. Thus, all link bandwidths in the Simgrid platform are divided by 0.92 to achieve the same theoretical bandwidth as in the simulations with ERS. This system does not directly provide the throughput of the system: we define its throughput as the number of processed instances divided by the total completion time. Unless stated, simulations only use the **Overlap** model, since it is a fundamental

Size (stages, processors)	Comp. times (seconds)	Comm. times (seconds)	#exp without critical resource / total
With overlap:			
(10, 20) and (10, 30)	5 . . . 15	5 . . . 15	0 / 220
(10, 20) and (10, 30)	10 . . . 1000	10 . . . 1000	0 / 220
(20, 30)	5 . . . 15	5 . . . 15	0 / 68
(20, 30)	10 . . . 1000	10 . . . 1000	0 / 68
(2, 7) and (3, 7)	1	5 . . . 10	0 / 1000
(2, 7) and (3, 7)	1	10 . . . 50	0 / 1000
Without overlap:			
(10, 20) and (10, 30)	5 . . . 15	5 . . . 15	14 / 220
(10, 20) and (10, 30)	10 . . . 1000	10 . . . 1000	0 / 220
(20, 30)	5 . . . 15	5 . . . 15	5 / 68
(20, 30)	10 . . . 1000	10 . . . 1000	0 / 68
(2, 7) and (3, 7)	1	5 . . . 10	10 / 1000
(2, 7) and (3, 7)	1	10 . . . 50	0 / 1000

Table 1: Numbers of experiments without critical resource.

assumption of most theorems. The whole code is available at `http://graal.ens-lyon.fr/~mgallet/downloads/SimuStochaGrid.tar.gz`.

7.1 Examples without any critical resources

In Section 4, we have shown examples of mappings without any critical resource, i.e., whose period is larger than any resource cycle-time, for both **Overlap** and **Strict** models. We have conducted extended experiments to assess whether such situations are very common or not. Several sets of applications and platforms were considered, with between 2 and 20 stages and between 7 and 30 processors. All relevant parameters (processor speeds, link bandwidths, number of processors computing the same stage) were randomly chosen uniformly within the ranges indicated in Table 1. Finally, each experiment was run for both models. We compared the inverse of the critical resource cycle-time and the actual throughput of the whole platform. A grand total of $5,152$ different experiments were run. Table 1 shows that the cases without critical resources are very rare, and the difference remains below 9%. In fact no such case was actually found with the **Overlap** model!

7.2 Number of processed data sets

As said in the previous paragraphs, the throughput is dependent of the number of processed data sets (Simgrid simulations) or of the number of simulated events (eg_sim simulations). Thus, we need to carefully assess this influence.

Figure 10 presents the throughput of a simulated system (7 stages, replicated 1, 3, 4, 5, 6, 7 and 1 times) using different numbers of processed data sets (resp. simulated events) and the theoretical throughput in the constant case. As we can see, the difference between exponential and constant cases is very small, and all measures tend to the same throughput. While the limit is reached in

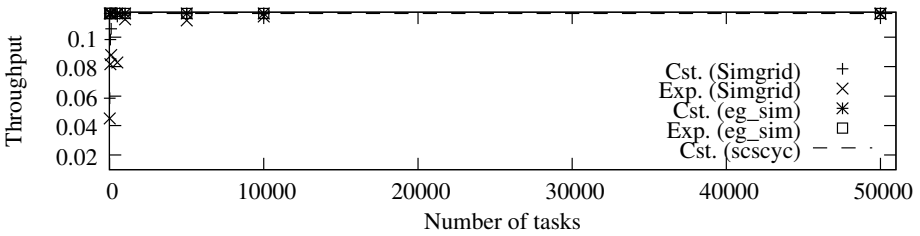

Figure 10: Variation of the throughput with the number of processed data sets.

almost cases as soon as 10,000 tasks (or events) are simulated, the Simgrid simulation with exponential times is a bit less stable but with 50,000 data sets, the difference with theoretical throughput is less than 1%.

7.3 Evaluation of the standard deviation in the exponential case

In this paper, we present several methods to compute the expectation of the throughput when communication and computation times are given by exponential random laws. However, we do not have any other information on this distribution. Thus, we used the same system to process 500 sets of 10, 50, 100, 500, 1,000, 5,000 or 10,000 data sets. This system is identical to the one used in Figure 10. Figure 11 displays the minimum, the maximum and the average throughput reached by the system after having processed 10, 50, 100, 500, 1,000, 5,000 or 10,000 data sets. Moreover, the standard deviation is also shown. Both the Simgrid and the eg_sim methods were used to determine the throughput.

As we can see, if the dispersion of Simgrid simulations is larger than the one of eg_sim simulations, it remains very small and the standard deviation is around 2% with 5,000 data sets and around 1% with 10,000 data sets. Thus, as expected by the small difference between constant and exponential cases, the variation is small and there is no need to use a lot of data sets to alleviate the effect of the random law.

7.4 Fidelity of the event graph model

Our event graph model clearly shows that the throughput can be determined by considering each stage and each communication between two successive stages as independent from other stages (resp. other communications).

We consider a system based on different numbers of a pattern of two stages linked by a single costly communication, Figure 12 represents the throughput of systems with different number of stages (using a pattern with 5 senders and 7 receivers, and $10,000$ data sets are used in these simulations). As expected, the throughput does not vary with the number of stages. This is due to the absence of backward dependences as explained in the event graph model: as soon as a data set is processed and pushed to the next step in the linear chain, it does not influence the previous stages anymore.

Thus, we consider a single communication to assess the difference between the Simgrid simulation and the different theoretical results. Since the complexity of the throughput evaluation stands in communications, we consider a single

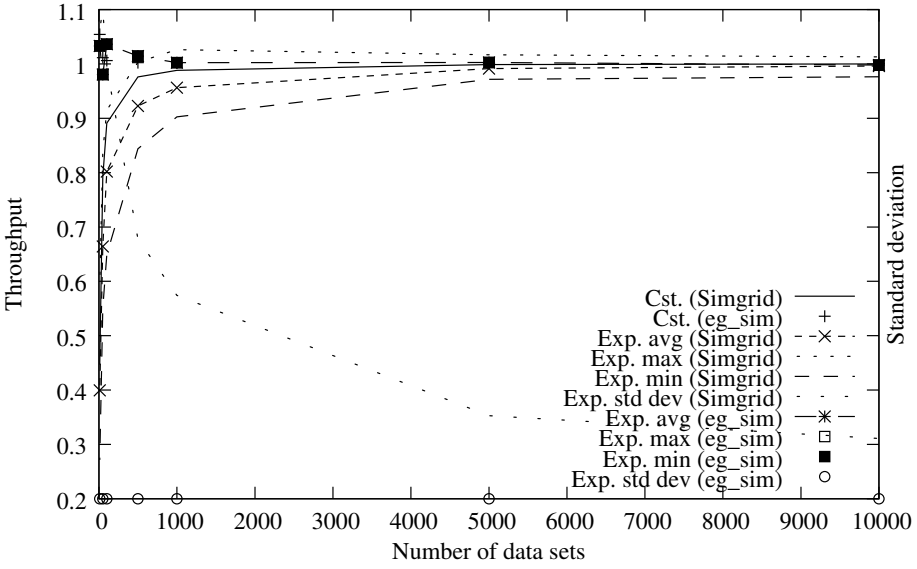

Figure 11: Minimum, maximum and average throughput across 500 simulation runs.

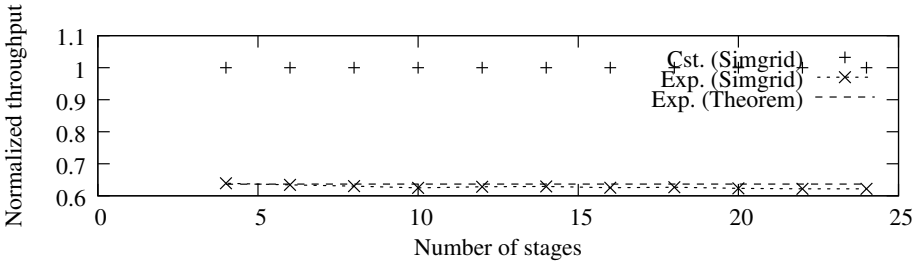

Figure 12: Variation of the throughput with the number of stages.

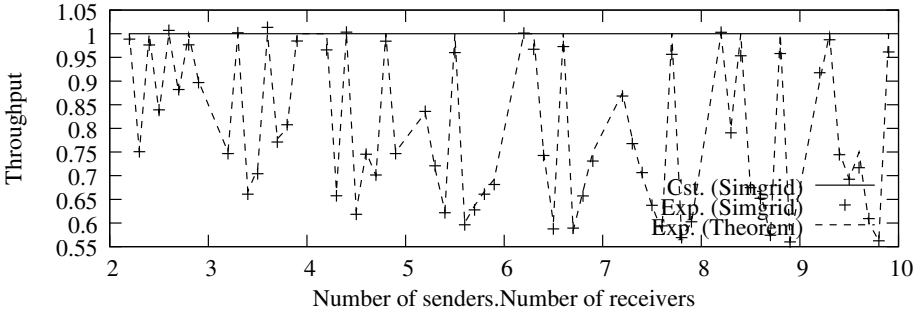

Figure 13: Throughput of constant and exponential cases, as predicted by Theorem 4 and reached by Simgrid simulations. All throughputs are normalized to the constant throughput.

communication between two negligible computations. Both stages have replication factors comprised between 2 and 9. Figure 13 displays the throughput reached by Simgrid simulations in both constant and exponential cases, and the throughput returned by Theorem 4 (only in the exponential case). As we can see, predicted values predicted are very close to those returned by Simgrid simulations.

In another simulation set, we still consider a single communication with negligible computations, but we assume a heterogeneous network. Thus, average communication time through a given link is randomly fixed between 100 and 1000, and all links have different mean communication times. As before, we evaluate the throughput with replication factors comprised between 2 and 9. Figure 14 shows the throughput reached by Simgrid and eg_sim simulations in both constant and exponential cases, and the theoretical throughput of the constant case, computed using the event graph model with scscyc. All throughputs are normalized to the throughput returned by Simgrid in the constant case.

We observe that the throughput of Simgrid and scscyc are identical, and all values are very close to the constant case; the difference is less than 2%. Unlike the homogeneous network case, there is almost no difference between the exponential and the constant case: due to the round-robin distribution, a single link limits all communications, and the behaviour tends to the behaviour of a communication through a single link.

7.5 Comparison between exponential and deterministic case

Figure 15 presents the difference between the expected throughput when using exponential random times (as reached by Simgrid simulations and as returned by Theorem 4) and the throughput returned by Simgrid simulations using constant times. If u is the number of senders and v is the number of receivers in the single involved communication, then the ratio between both cases is equal to $1 \geq \frac{\max(u,v)}{u+v-1} > \frac{1}{2}$.

Again, this figure clearly shows the correlation between throughput achieved by Simgrid simulations and predicted throughput.

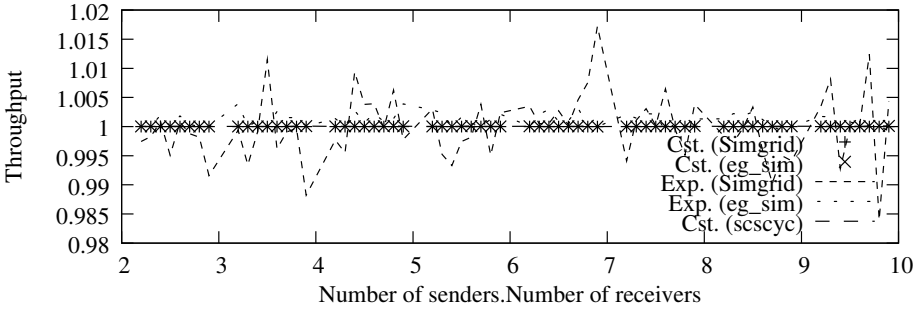

Figure 14: Throughput of constant and exponential cases and reached by Simgrid simulations. All throughputs are normalized to the constant throughput.

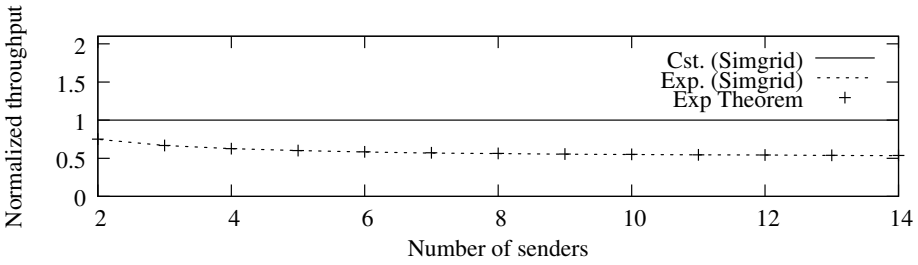

Figure 15: Comparison between constant and exponential laws.

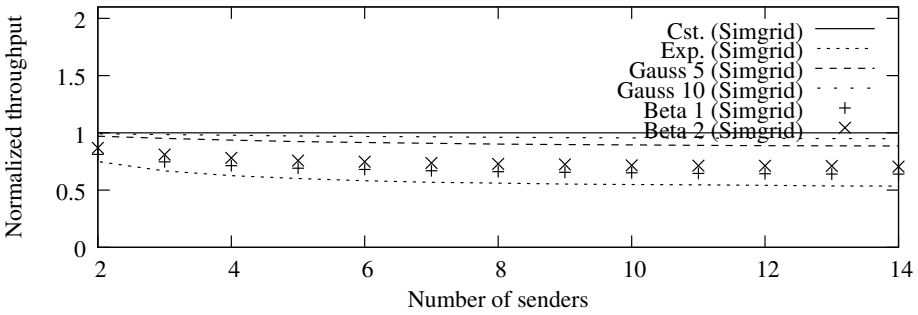

Figure 16: Comparison of several NBUE laws.

7.6 Comparison of several random laws

Figure 16 presents the throughput reached by several other NBUE random laws. As predicted by Theorem 7, their throughput is comprised between the throughput reached by constant times and the one reached by exponential times. In this experiment, Gauss X means a normal distribution with variance $\sqrt{X}$, and Beta X means a beta distribution of shape X. The mean value is the same for all distributions.

Figure 17 presents the throughput reached by several non-NBUE random laws. As we can see, the resulting expected throughput can by either larger or smaller than the throughput with constant or exponential times.

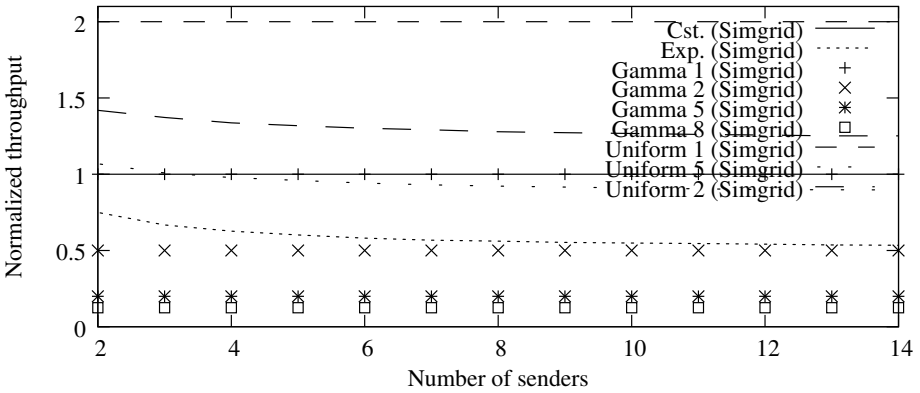

Figure 17: Comparison of several non-NBUE laws.

7.7 Running time of simulations

All tools (scscyc, eg_sim and Simgrid simulator) are coded in C and are quite fast: less than one second is sufficient to generate all tasks and run all tools with 100 data sets (for Simgrid simulator) and 100 events (for eg_sim). Event with 100,000 events and tasks, the whole simulation takes around three minutes.

8 Conclusion

In this paper, we have investigated how to compute the throughput achieved by a given one-to-many mapping of a streaming application onto a target heterogeneous platform. The major novelty is the introduction of I.I.D. variables to model computation and communication times. Our model is based on event graphs and Markov chains, and allows to compute the expected throughput when variables are constant or follow exponential random laws.

In the general case of arbitrary I.I.D. and N.B.U.E. random variables, we have established bounds, and the lower and upper bounds are obtained by the deterministic and exponential cases respectively. Both bounds can be computed in polynomial time under the **Overlap** model with a homogeneous communication network.

Now that we have new methods to evaluate the throughput of a given mapping in a probabilistic setting, we will devote future work to designing polynomial time heuristics for the NP-complete problem mentioned above. Thanks to the methodology introduced in this paper, we will be able to compute the throughput of heuristics and compare them together. This would be a first and important step in the field of scheduling streaming applications on large-scale platforms whose load and performance are subject to dynamic variations.

Acknowledgment

Part of this work has appeared in ICPP'09 and SPAA'10. Anne Benoit and Yves Robert are with the Institut Universitaire de France. This work was supported in part by the ANR StochaGrid project and by the Inria ALEAE project.

References

[1] F. Baccelli, G. Cohen, and B. Gaujal. Recursive Equations and Basic Properties of Timed Petri Nets. *Journal of Discrete Event Dynamic Systems*, 1(4), 1992.

[2] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[3] A. Benoit, F. Dufossé, M. Gallet, B. Gaujal, and Y. Robert. Computing the throughput of probabilistic and replicated streaming applications. In *22nd ACM Symposium on Parallelism in Algorithms and Architectures SPAA 2010*. ACM Press, 2010.

[4] A. Benoit, M. Gallet, B. Gaujal, and Y. Robert. Computing the throughput of replicated workflows on heterogeneous platforms. In *ICPP'2009*, 2009.

[5] A. Benoit and Y. Robert. Mapping pipeline skeletons onto heterogeneous platforms. *J. Parallel Distributed Computing*, 68(6):790–808, 2008.

[6] M. D. Beynon, T. Kurc, A. Sussman, and J. Saltz. Optimizing execution of component-based applications using group instances. *Future Generation Computer Systems*, 18(4):435–448, 2002.

[7] H. Casanova, A. Legrand, and M. Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *ICCMS'08*, Mar. 2008.

[8] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation*, 24(1-2):47–68, 1995.

[9] M. Cole. Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. *Parallel Computing*, 30(3):389–406, 2004.

[10] DataCutter Project: Middleware for Filtering Large Archival Scientific Datasets in a Grid Environment. http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm.

[11] B. Gaujal and J.-M. Vincent. *Introduction to Scheduling*, chapter Comparisons of stochastic task-resource systems. CC Press, 2009.

[12] O. Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.

[13] H. Hillion and J.-M. Proth. Performance evaluation of job shop systems using timed event graphs. *IEEE TAC*, 34(1):3–9, 1989.

[14] A. Jean-Marie. ERS: a tool set for performance evaluation of discrete event systems. http://www-sop.inria.fr/mistral/soft/ers.html.

[15] J. Kamburowski. Bounding the distribution of project duration in pert networks. *Operation Research Letters*, 12:17–22, 1992.

[16] D. E. Knuth. *The Art of Computer Programming. Volume 3, second edition*. Addison-Wesley, 1998.

[17] Y. Kumazawa. Tests for new better than used in expectation with randomly censored data. *Sequen.Anal.*, 5:85–92, 1986.

[18] M. Spencer, R. Ferreira, M. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz. Executing multiple pipelined data analysis operations in the grid. In *Supercomputing'02*. ACM Press, 2002.

[19] J. Subhlok and G. Vondran. Optimal mapping of sequences of data parallel tasks. In *PPoPP'95*, pages 134–143. ACM Press, 1995.

[20] J. Subhlok and G. Vondran. Optimal latency-throughput tradeoffs for data parallel pipelines. In *SPAA'96*, pages 62–71. ACM Press, 1996.

[21] K. Taura and A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *HCW'00*, pages 102–115. IEEE Computer Society Press, 2000.

[22] P. Velho and A. Legrand. Accuracy study and improvement of network simulation in the simgrid framework. In *Simutools '09*, pages 1–10. ICST, 2009.

[23] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz. Toward optimizing latency under throughput constraints for application workflows on clusters. In *Euro-Par'07*, LNCS 4641, pages 173–183. Springer Verlag, 2007.

[24] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz. A duplication based algorithm for optimizing latency under throughput constraints for streaming workflows. In *ICPP'2008*, pages 254–261. IEEE Computer Society Press, 2008.

[25] Q. Wu and Y. Gu. Supporting distributed application workflows in heterogeneous computing environments. In *ICPADS'08*. IEEE Computer Society Press, 2008.

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