Computing the throughput of probabilistic and replicated streaming applications*

(Regular Submission)

A. Benoit[1,*], F. Dufossé[1], M. Gallet[1], B. Gaujal[2], Y. Robert[1]

1. LIP (jointly operated by CNRS, ENS Lyon, INRIA and UCB Lyon), ENS Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France.
{Anne.Benoit|Fanny.Dufosse|Matthieu.Gallet|Yves.Robert}@ens-lyon.fr
2. LIG (jointly operated by CNRS, Grenoble INP, INRIA, UJF and UPMF), Grenoble, France.
Bruno.Gaujal@imag.fr

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 (although of exponential cost) to compute the throughput when mapping parameters follow I.I.D. exponential laws. This general method is based upon the analysis of timed Petri nets deduced from the application mapping; it turns out that these 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 of the paper 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.

Keywords: scheduling, probabilistic streaming applications, replication, throughput, timed Petri nets.

*** Corresponding author –** Anne Benoit
LIP, ENS Lyon, 46 Allée d'Italie, 69364 Lyon Cedex 07, France
Phone: (+33) 4 72 72 87 58
Email: Anne.Benoit@ens-lyon.fr

*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.

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 [15, 14, 18]. They are a popular programming paradigm for streaming applications like video and audio encoding and decoding, DSP applications, etc [6, 17, 20]. 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 [15, 16] and of the DataCutter team [4, 14, 19]. This also corresponds to the *dealable* stages of Cole [5]. 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 compute and communicate in parallel.

The major novelty of the paper is to 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 events (computations or communications) that occur on the same hardware resource (processor or 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 different CPU and data volumes; furthermore, the distribution law may well vary from one computation to another, or from one communication to another. To the best of our knowledge, this paper is the first attempt to compute the throughput of a mapping whose parameters obey probability distribution laws.

In the deterministic case, and without replication, the throughput of a given mapping is easily seen to the 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. This result was shown in [2], using a representation of the mapping based on timed Petri nets. In this paper we build upon the latter construction to tackle the probabilistic case.

The first 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 of the paper 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.

Our last contribution departs from the main trend of the paper and deals with the problem of determining the optimal mapping, i.e., the one-to-many mapping that optimizes the throughput. Indeed, the optimal mapping could enjoy a particular structure that renders the computation of the throughput easier than for an arbitrary given mapping. It could even be polynomial for arbitrary laws and both the **Strict** and OVERLAP models! We prove that this is in fact not the case: determining the optimal mapping is NP-complete, even in the deterministic case and without any communication cost (hence for both models).

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 explain how to compute the throughput when communication and computation times follow I.I.D. exponential laws (Section 3). 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 4, we deal with arbitrary I.I.D. and N.B.U.E. laws, and we establish the above-mentioned bounds on the throughput. We assess the NP-completeness of the mapping optimization problem in Section 5. Finally, we present some conclusions and directions for future work in Section 6.

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 Before moving to the probabilistic study, we recall existing results for the deterministic case 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 m 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_m gathers the final data.

The workflow is executed on a fully heterogeneous platform with M processors. The speed of processor P_p is denoted as s_p (in flops). We assume bidirectional links $\text{link}_{p,q} : P_p \rightarrow P_q$ 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.

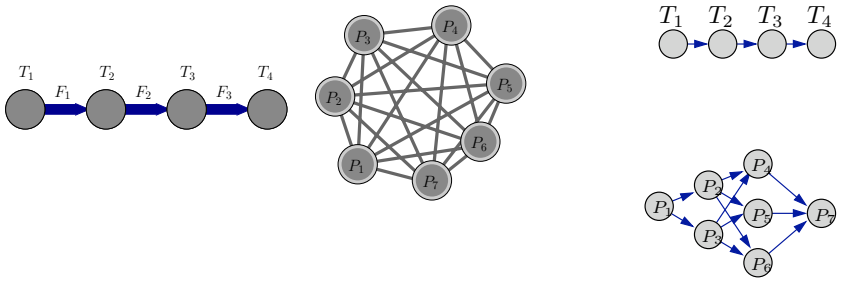

Figure 1: Example A: Four-stage pipeline, seven-processor computing platform, mapping with replication.

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* [3], 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* [15, 16, 4, 14, 19] or *dealable* [5].

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 for previous data sets, hence the possibility to process different data sets in different locations. The following scheme illustrates the replication of a stage T_k onto three processors:

$$\ldots\ T_{k-1} \begin{array}{ll} \diagup & T_k \text{ on } P_1\text{: data sets } \mathbf{1, 4, 7, \ldots} \\ -- & T_k \text{ on } P_2\text{: data sets } \mathbf{2, 5, 8, \ldots} \\ \diagdown & T_k \text{ on } P_3\text{: data sets } \mathbf{3, 6, 9, \ldots} \end{array} \begin{array}{l} \diagdown \\ -- \\ \diagup \end{array} T_{k+1}\ \ldots$$

For $1 \leq k \leq N$, let R_k denote the number of processors participating to the processing of T_k. For $1 \leq p \leq M$, if P_p participates to the work of T_k, then we write $p \in \mathit{Team}_k$ and define $R'_p = R_k$. 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 on the non-replicated stage. 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_k. Then, if $p \in \mathit{Team}_k$, P_p processes one data set every R_k data sets at the speed dictated by P_{slow}, and thus its computation time (per data set) is $C_{\text{comp}}(p) = \frac{w_k}{R_k \times s_{slow}}$. 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 Computing the throughput in the deterministic case

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, 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. Here are the main results that we established previously in [2]:

- Model **Overlap**: the throughput can be determined in polynomial time.
- Model **Strict**: determining the complexity of this problem remains an open question. However, the throughput can be computed in time $O\left(\text{lcm}_{1 \leq i \leq N}(R_i)^3\right)$, a possibly exponential resolution time.

In the following, we investigate how to compute the throughput when execution and communication times are subject to random variations.

2.4 Random variables

We consider in the following that the time to execute a stage, and the time to transfer data, are random variables. Thus, in the deterministic case, we can denote the n-th computation time of stage T_i on processor P_p by $c_p = w_i/s_p$. Similarly, the n-th communication time of the file F_i sent by P_p to P_q is given by $d_{p,q} = \delta_i/b_{p,q}$.

Let $X_p(n)$ be the random variable giving the actual computation time of the n-th data set processed by P_p, where $p \in \textit{Team}_k$ (recall that each processor deals with only one stage). In the deterministic case, we have $X_p(n) = w_k/s_p$ for all n, but in the probabilistic setting the $X_p(n)$ will be random variables obeying I.I.D. laws. Similarly, let $Y_{p,q}(n)$ be the random variable giving the actual communication time of the n-th file of type F_k transferred from P_p to P_q, where $p \in \textit{Team}_k$ and $q \in \textit{Team}_{k+1}$. In the deterministic case, we have $Y_{p,q}(n) = \delta_k/b_{p,q}$ for all n. Again, in the probabilistic setting, the $Y_{p,q}(n)$ will be random variables obeying I.I.D. laws. Finally, we let (X, Y) denote the mapping of an application $(T_1, \ldots, T_N)$ on a platform $(P_1, \ldots, P_M)$.

The probability that the computation time of T_i on P_p is larger than x is given by $\Pr(X_p(n) > x)$, while its expected value is given by $\mathbf{E}[X_p(n)]$. This definition is general and does not imply any special constraint on the involved random variables. However, some of our results are only valid for specific classes 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. ([13]).

Independent and identically-distributed variables. A collection of random variables is said to be independent and identically-distributed (I.I.D.) if each random variable has the same probability distribution as the others and all variables are mutually independent. This assumption will hold true throughout the

paper: processing times $\{X_p(n)\}_{1\leq p\leq M, n\in\mathbb{N}}$ and communication times $\{Y_{p,q}(n)\}_{1\leq p,q\leq M, n\in\mathbb{N}}$ always are independent I.I.D. sequences.

3 Computing the throughput with exponential laws

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 4), exponential laws are extreme cases among all N.B.U.E. variables.

In the rest of this section, we first recall the design principles of the timed Petri net model (Section 3.1). Then in Section 3.2, we 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 3.3). Finally in Section 3.4, we derive a polynomial algorithm for the **Overlap** case when we further assume that the communication network is homogeneous.

3.1 Timed Petri nets

As in [2], we model the system formed by a given mapping (of an application onto a platform) by a timed Petri net. Here we briefly recall design principles:

- any path followed by the input data sets is fully developed into the timed Petri net (as a row); there are $R = \mathrm{lcm}_{1\leq i\leq N}(R_i)$ such rows;
- any computation or communication is represented by a transition, whose firing time is the same as the computation time (respectively communication time); it appears on a column of the timed Petri net, which has $2N-1$ columns;
- dependencies between two successive operations are represented by places between the transitions corresponding to these operations.

The complete timed Petri nets representing Example A (Figure 1) with the **Overlap** and **Strict** models are shown in Figure 2. There are four processing stages, thus three communications and a total of seven columns. In the example, stages 1 and 4 are not replicated, while T_2 (resp. T_3) is replicated onto two (resp. three) processors, as depicted in Figure 1. There are thus $2\times 3 = 6$ lines in both Petri nets. The dependencies depend upon the model (**Overlap** or **Strict**), and they enforce the round-robin distribution of data sets. Note that in all cases, the timed Petri net is an event graph (each place has a single input transition and a single output transition).

3.2 General method to compute the throughput

Theorem 1. *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 $O\left(\exp(\mathrm{lcm}_{1\leq i\leq N}(R_i))^3\right)$.*

We only present here the main steps of the complete proof, and the detailed proof can be found in Appendix A:

1. model the system by a timed Petri net;
2. transform this timed Petri net into a Markov chain;
3. compute the stationary measure of this Markov chain;

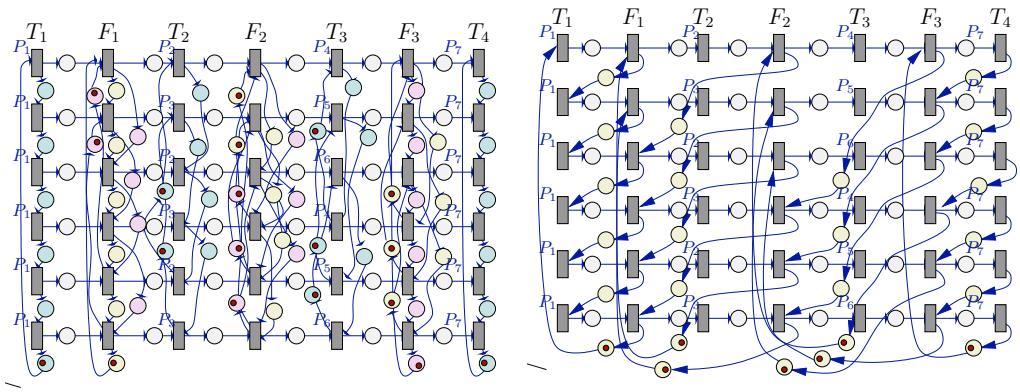

Figure 2: Timed Petri net representing Example A, **Overlap** and **Strict** model.

4. derive the throughput from the marginals of the stationary measure.

3.3 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 2. *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*

$$O\left(N \exp(\max_{1 \leq i \leq N}(R_i))^3\right).$$

Here again, due to a lack of space, we only give the overall structure of the proof, while we refer to Appendix B for a detailed 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 $u_j \times v_j$;
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 [12];
7. compute the throughput of $\mathcal{P}_j$ in isolation (called inner throughput of component D_j in the following);
8. combine the inner throughputs of all components to get the global throughput of the system.

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. This allows us to decrease the overall complexity of the computation, similarly to the idea used in [2].

3.4 Overlap model, homogeneous communication network

In the case where all the communications 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 3. *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.*

In the process of proving Theorem 3 (see Appendix C), we establish several interesting properties. First, we prove that the inner throughput of a processor component D_j is $\rho_j = \lambda_j$. Then, for a communication strongly connected component D_j, which is made of copies of a same pattern of size $u_j \times v_j$, we prove that its inner throughput is equal to $\rho_j = \frac{\lambda_j}{v_j+u_j-1}$. This latter value, obtained in the exponential case, has to be compared with the throughput in the deterministic case where all communication times are deterministic ($1/\lambda_j$), which is equal to $\rho_j = \frac{\lambda_j}{\max(v_j,u_j)}$. The fact that the throughput in the exponential case is lower than the throughput in the deterministic case will be explained in Section 4. However, the fact that the throughput can be computed explicitly is much more unexpected, since such explicit formulas are known to be very hard to obtain, even for simple event graphs [1].

The global throughput can then be computed in polynomial time from all inner throughputs, and it is equal to:

$$\rho = \sum_{D_j \in C_{2N-1}} \min_{D_{j'} \prec D_j} \rho_{j'}, \tag{1}$$

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 components 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).

The detailed proof of this theorem can be found in Appendix C. It starts similarly to the proof of Theorem 2, but the computation of inner throughputs is simplified, since we have explicit formulas in all cases (as explained above).

4 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 [11]. 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 4.1). Moreover, the lower and upper bounds are obtained by the deterministic and exponential cases respectively.

4.1 Theoretical comparison

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

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 4. *Consider two systems $(X^{(1)}, Y^{(1)})$ and $(X^{(2)}, Y^{(2)})$. If $\forall 1 \leq p \leq M, X_p^{(1)}(n) \leq_{\mathrm{st}} X_p^{(2)}(n)$ and $\forall 1 \leq p, q \leq M, Y_{p,q}^{(1)}(n) \leq_{\mathrm{st}} Y_{p,q}^{(2)}(n)$, 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: $D^b(n) = D^b(n-1) \otimes A^b(n)$ [1], 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 $D^1(n) \leq_{\mathrm{st}} D^2(n)$.

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 5. *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, $\forall 1 \leq p \leq M, X_p^{(1)}(n) \leq_{\mathrm{icx}} X_p^{(2)}(n)$ and $\forall 1 \leq p, q \leq M, Y_{p,q}^{(1)}(n) \leq_{\mathrm{icx}} Y_{p,q}^{(2)}(n)$, 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. □

Theorem 6. *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 $\mathbf{E}[X_p^{(1)}(n)] = \mathbf{E}[X_p^{(2)}(n)]$,*
- *$\forall 1 \leq p, q \leq M, Y_{p,q}^{(2)}(n)$ has an exponential distribution, and $\mathbf{E}[Y_{p,q}^{(1)}(n)] = \mathbf{E}[Y_{p,q}^{(2)}(n)]$,*
- *$\forall 1 \leq p \leq M, X_p^{(3)}(n)$ is deterministic and for all n, $X_p^{(3)}(n) = \mathbf{E}[X_p^{(2)}(n)]$,*
- *$\forall 1 \leq p, q \leq M, Y_{p,q}^{(3)}(n)$ is deterministic and for all n, $Y_{p,q}^{(3)}(n) = \mathbf{E}[Y_{p,q}^{(2)}(n)]$.*

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 [8], 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 5 shows that $\rho^{(3)} \geq \rho^{(1)} \geq \rho^{(2)}$. □

In particular, Theorem 6 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 (given by Formula (1), where the inner throughput of processing components are the same as in the exponential case and the throughput of communication components is replaced by $\frac{\lambda_i}{\max(u_i, v_i)}$ and the throughput of the system in which all random processing times are replaced by exponential variables with the same mean value, given by Formula (1).

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

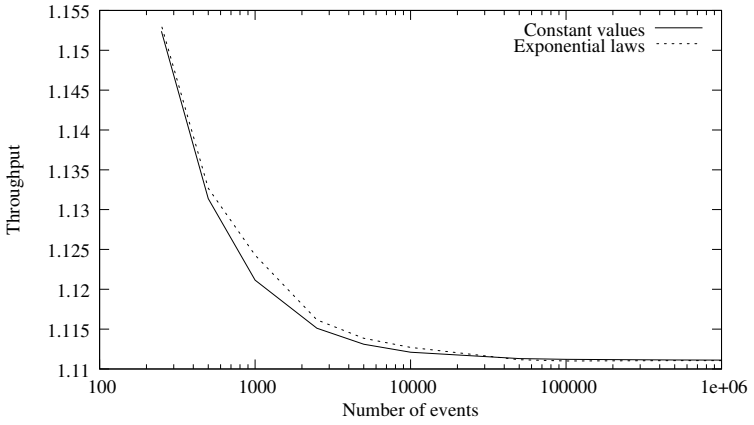

Figure 3: Evolution of the measured throughput with the number of samples.

Distribution	Constant value c	Exponential mean c	Uniform between $c/2$ and $3c/2$	Uniform between $c/10$ and $19c/10$	Pareto mean c
Throughput	2.0299	2.0314	2.0304	2.0305	2.0300

Table 1: Throughput obtained with several distributions of same mean.

4.2 Numerical experiments

In this section, we compare the behavior of several random distributions of same mean for an application made of $N = 8$ stages, and a 31-processor platform. The expected throughput is numerically determined using the ERS software [10], by simulation up to convergence to the stationary behavior. Figure 3 shows that at least 100,000 samples are required to reach stable values. We compare constant, exponential, uniform and Pareto distributions. As can be seen in Table 1, all throughputs are comprised between the throughput of the deterministic system and the one with exponential laws, in accordance with Theorem 6 for uniform distributions that are NBUE. Also note that the bounds still hold for the Pareto law even though it is not N.B.U.E.. Moreover, these figures are quite close, which is good news: replacing a random variable by either a constant value or by an exponential law with same mean may well lead to very good approximations.

5 Finding the optimal mapping

In this section, we depart from the main trend of the paper in that we do not consider that the mapping of the application onto the platform is given. Instead, we aim at determining the optimal mapping, i.e., the one-to-many mapping that optimizes the throughput. The rationale is that the optimal mapping could enjoy a particular structure that would facilitate renders the computation of the throughput. In other words, computing the *optimal* throughput might be easier than computing the throughput of an arbitrary mapping We prove that this is not the case: determining the optimal mapping is NP-complete, even in the deterministic case and without any communication cost (hence for both models). Note that the one-to-one problem, i.e. without replication, was shown to have polynomial complexity in [3]

Theorem 7. *In the deterministic case, the problem of finding the one-to-many mapping (with replication) which minimizes the period on a heterogeneous platform without communication costs, is NP-complete.*

Proof. Consider the associated decision problem: given a period K, does there a mapping whose period

does not exceed K? The problem is obviously in NP: given a period and a mapping, it is easy to check in polynomial time whether it is valid or not.

The NP-completeness is obtained by a reduction from 3-PARTITION, which is NP-complete in the strong sense [7]. Let $\mathcal{I}_1$ be an instance of 3-PARTITION: given a set $A = \{a_1, ..., a_{3m}\}$ and an integer B, with $\forall 1 \leq i \leq 3m$, $\frac{B}{4} < a_i < \frac{B}{2}$ and $\sum_{1 \leq i \leq 3m} a_i = mB$, does it exist m disjoint subsets $A_1, ..., A_m$ of A such that $\forall 1 \leq j \leq m, \sum_{a_i \in A_j} a_i = B$? We construct an instance $\mathcal{I}_2$ of our problem with $3m$ pipeline stages and $\frac{m(m+1)}{2}B$ processors such that:

- $\forall 1 \leq k \leq 3m, w_k = m! \times a_k$ (computation cost of stage T_k);
- $\forall 1 \leq j \leq m$, there are exactly $j \times B$ processors of speed $\frac{m!}{j}$;
- the period is fixed to $K = 1$.

Note that in this instance the sum of the speeds of all processors is equal to the sum of computation costs of all stages. This proves that in a mapping of period 1, processors cannot be idle. Therefore, all processors allocated to a same stage must have the same speed (see Section 2.2). Also, since 3-PARTITION is NP-complete in the strong sense, we can encode $\mathcal{I}_1$ in unary. Then, the values in $\mathcal{I}_2$ (stage computation costs, processor speeds, period) can be encoded in binary and thus their size is polynomial in the size of $\mathcal{I}_1$.

Now we show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ has a solution. Suppose first that $\mathcal{I}_1$ has a solution $A_1, ..., A_m$. For all $1 \leq j \leq m$, for all i such that $a_i \in A_j$, we associate the stage T_i of computation cost w_i to $a_i \times j$ processors of speed $\frac{m!}{j}$. Since $\sum_{a_i \in A_j} a_i \times j = B \times j$, this solution respects the number of available processors. We obtain, for all $1 \leq i \leq 3m$ such that $a_i \in A_j$, a period $\frac{a_i \times j}{a_i \times j} = 1$. This proves that this mapping is a valid solution for $\mathcal{I}_2$.

Suppose now that $\mathcal{I}_2$ has a solution. We know that all processors allocated to a given stage have same speeds, otherwise the period would be greater than 1. For $1 \leq j \leq m$, let A_j be the set of a_k such that stage T_k is mapped onto processors of speed $\frac{m!}{j}$. We obtain $\forall j, \sum_{a_k \in A_j} a_k \times m! \leq j \times B \frac{m!}{j}$, which means $\forall j, \sum_{a_k \in A_j} a_k \leq B$. Since we have $\sum_{1 \leq i \leq 3m} a_i = mB$, we derive that $\forall j, \sum_{a_k \in A_j} a_k = B$. Therefore, $A_1, ..., A_m$ is a solution for $\mathcal{I}_1$. This concludes the proof. □

6 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. In previous work [2], we have introduced methods to compute the throughput in the deterministic case, using timed Petri nets. We extended these results to the situation where computation and communication times follow I.I.D. exponential laws, providing a method whose cost may be exponential. We have refined this result and derived a polynomial-time algorithm for the **Overlap** model and a homogeneous communication network. able to compute the throughput explicitly. In the general case of arbitrary I.I.D. and N.B.U.E. random variables, we have established bounds on the 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. We also proved that determining the mapping that maximizes the throughput is an NP-complete problem, even in the simpler deterministic case with no communication costs.

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.

References

[1] F. Baccelli and D. Hong. Analyticity of iterates of random non-expansive maps. Research Report 3558, INRIA, Sophia-Antipolis, 1998.

[2] A. Benoit, M. Gallet, B. Gaujal, and Y. Robert. Computing the throughput of replicated workflows on heterogeneous platforms. In *Proceedings of 38th International Conference of Parallel Processing*, 2009.

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

[4] 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.

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

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

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

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

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

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

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

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

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

[14] 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.

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

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

[17] 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.

[18] 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: Parallel Processing*, LNCS 4641, pages 173–183. Springer Verlag, 2007.

[19] 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.

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

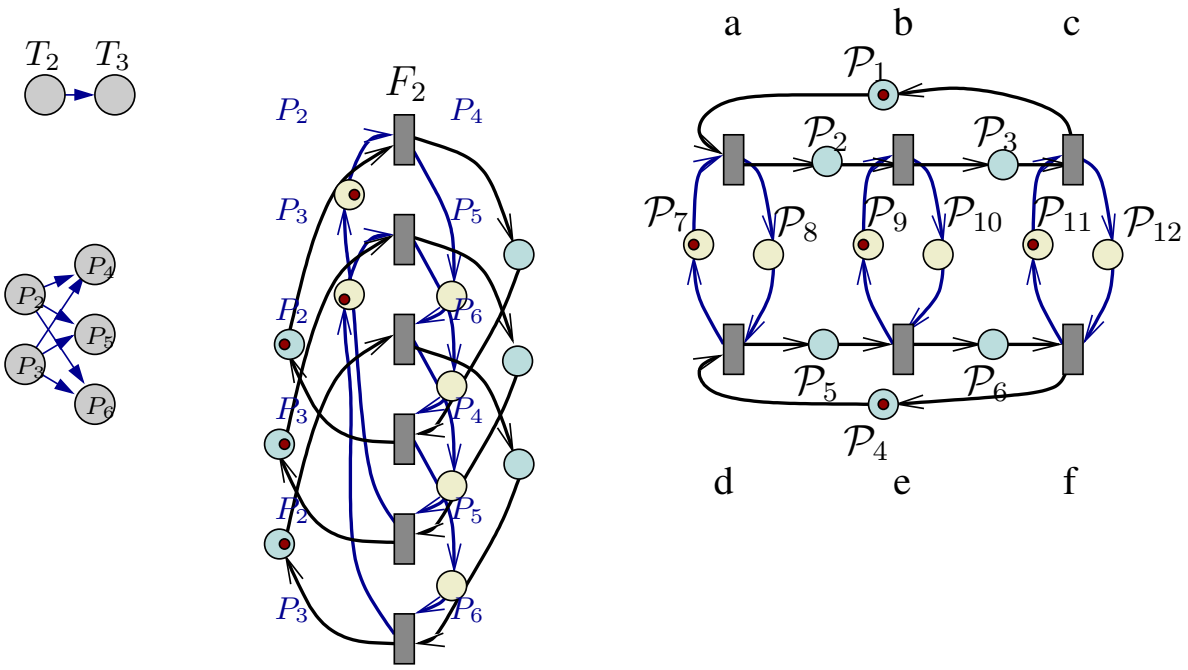

Figure 4: Example A: Part of the timed Petri net corresponding to communication F_2.

A Proof of Theorem 1

Proof. First, we present the main steps of the complete proof:

1. model the system by a timed Petri net;
2. transform this timed Petri net into a Markov chain;
3. compute the stationary measure of this Markov chain;
4. derive the throughput from the marginals of the stationary measure.

Model the system as a timed Petri net. As said before, the transformation of the initial system into a timed Petri net is fully described in [2] and we do not detail it entirely here. Recall from Section 3.1 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. 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 $\mathcal{M}_i$ 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 representation of the Markov chain

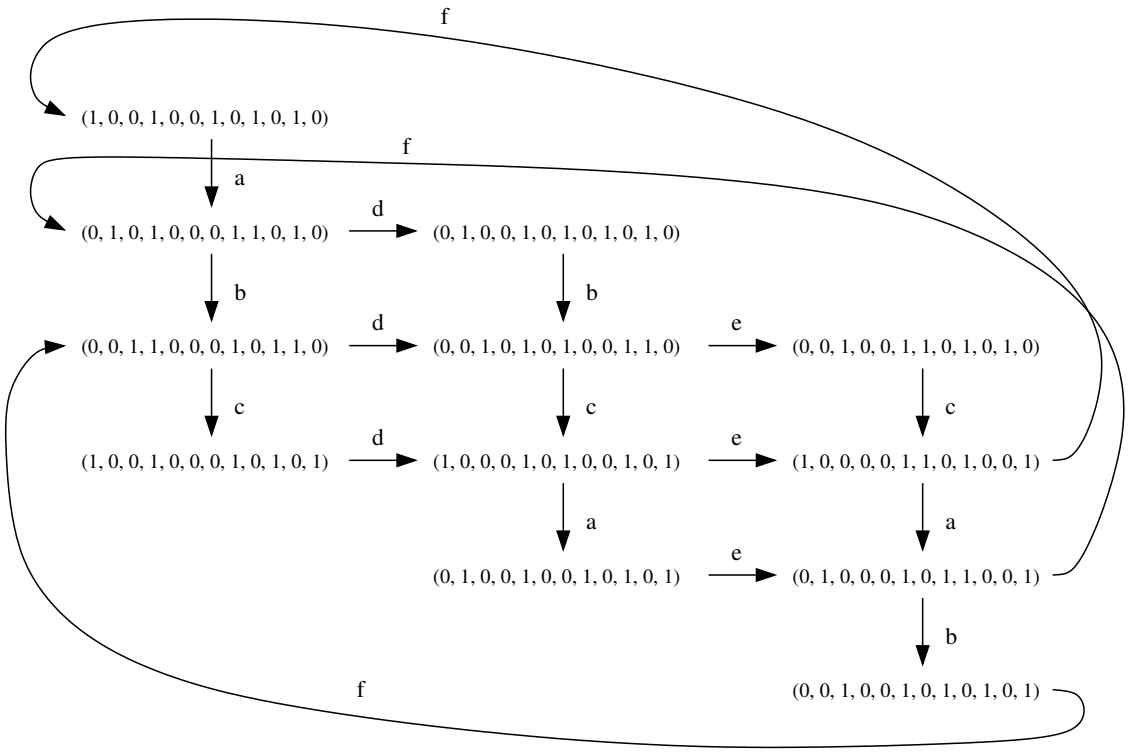

Figure 5: List of all possible states of the Markov chain corresponding to the reduced timed Petri net of Example A.

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 [9]. The sum of the probability of the valid states returns the throughput.

□

B Proof of Theorem 2

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 $u_j \times v_j$;
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 [12];
7. compute the throughput of $\mathcal{P}_j$ in isolation (called inner throughput of component D_j in the following);
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 [2]: 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 firing 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 (column C_{2i}): it involves R_i senders and R_{i+1} receivers. We already know that the timed Petri net is made of $g = \gcd(R_i, R_{i+1})$ connected components. Let u be equal to R_i/g and v 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 B 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, 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 6. As said before, determining the throughput of $\mathcal{P}$ is equivalent to determine a stationary measure of a Markov chain. We know that the stationary measure of a Markov chain with t states can be computed in time $O\left(t^3\right)$ [9]. 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 $\mathcal{P}_j$ of Figure 6 is represented in Figure 7. 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 7. 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 8). If i and j are given, then there are $\alpha_{i,j} = \begin{pmatrix} i+j \\ i \end{pmatrix}$ possible paths from $(i, 0)$ to $(0, j)$, where $\begin{pmatrix} n \\ k \end{pmatrix}$ 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:

$$\begin{aligned} S(u,v) &= \sum_{i=0}^{u-1} \sum_{j=0}^{v-1} \alpha_{i,j}\alpha_{u-1-i,v-1-j} \\ &= \sum_{i=0}^{u-1} \sum_{j=0}^{v-1} \begin{pmatrix} i+j \\ i \end{pmatrix} \begin{pmatrix} u+v-2-i-j \\ u-1-i \end{pmatrix} \\ &= \begin{pmatrix} u+v-1 \\ u-1 \end{pmatrix} v = \tfrac{(u+v-1)!}{(u-1)!v!} v \end{aligned}$$

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 $O\left(N \exp(\max_{1 \leq i \leq N}(R_i))^3\right)$.

□

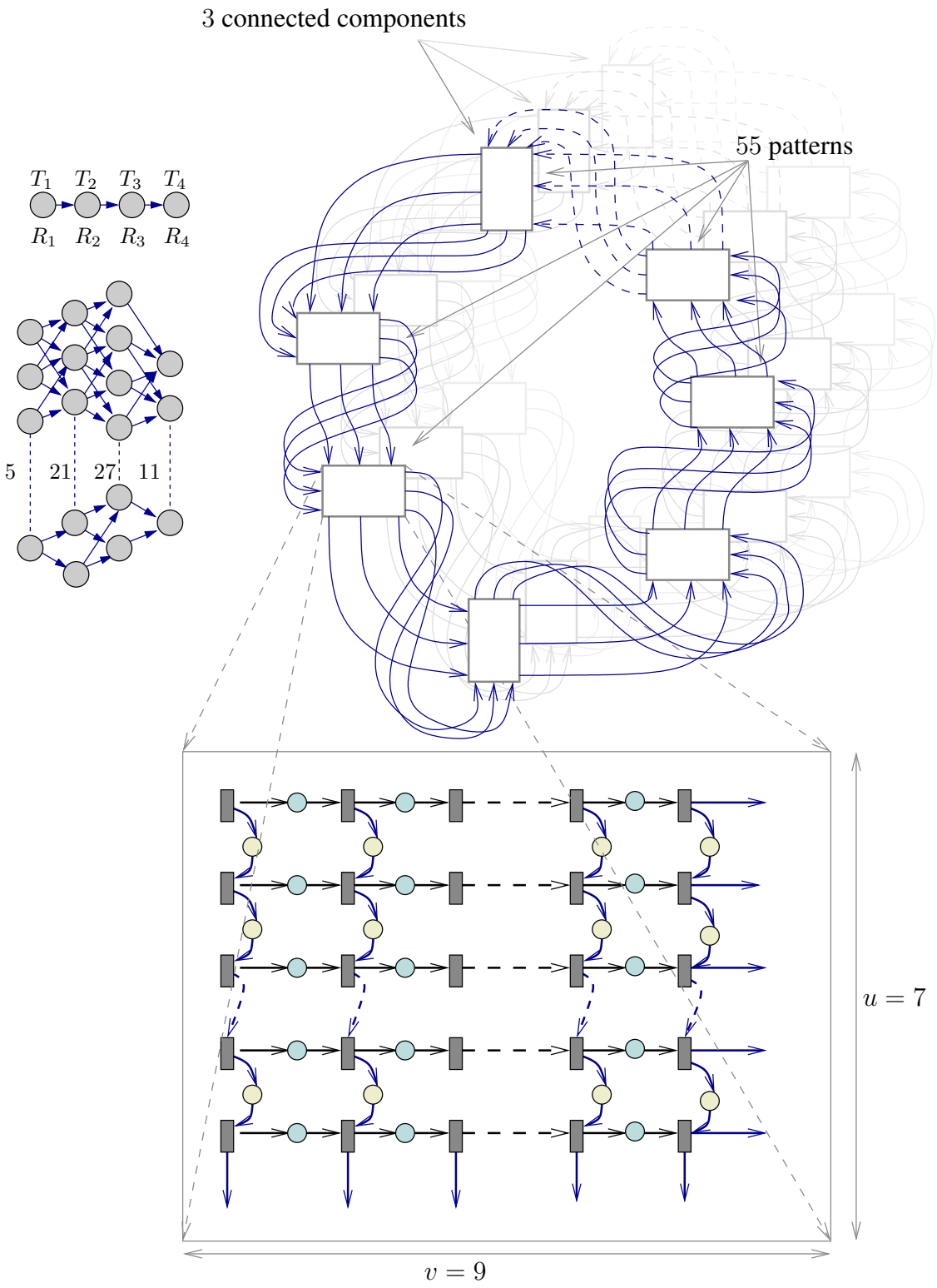

Figure 6: Example B, with stages replicated on 5, 21, 27 and 11 processors, and structure of the timed Petri net corresponding to the second communication.

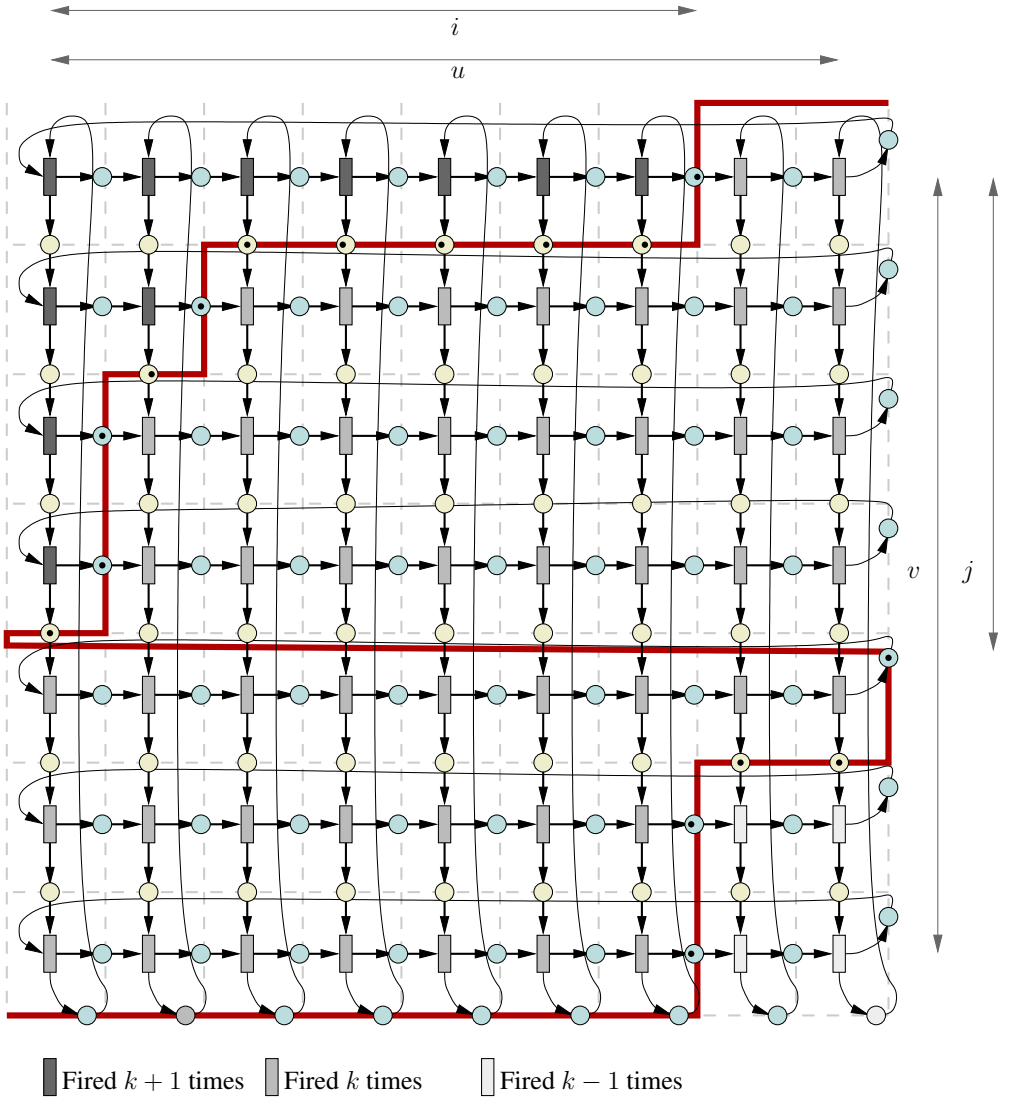

Figure 7: Valid marking of $\mathcal{P}_j$, the reduced timed Petri net of the second communication of Example B.

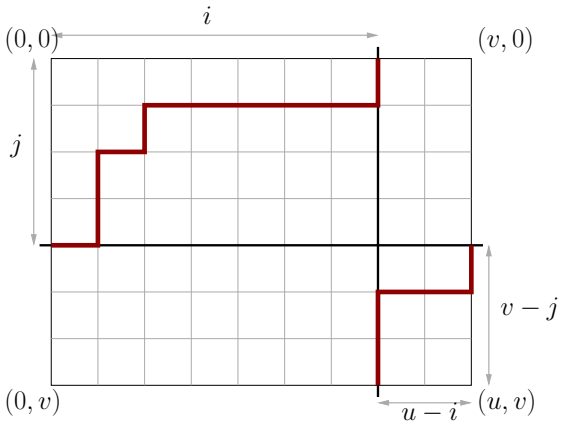

Figure 8: Representation with Young diagrams of a valid marking.

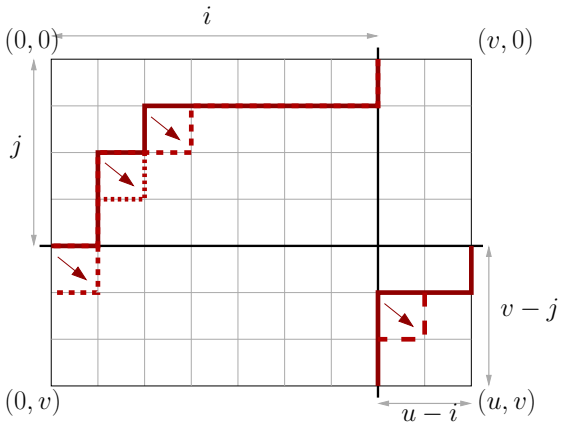

Figure 9: Reachable states from a given position.

C Proof of Theorem 3

Proof. Platforms with the **Overlap** model and a homogeneous communication network are special cases of the **Overlap** model. Thus, the demonstration of Theorem 2 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 7, 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 7 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 [9]: 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 2):

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

Finally, we know the rate of the states leading to a given transition, and the number of states leading to it. Thus, the inner throughput is equal to $\frac{\lambda_j}{(v+u-1)}$.

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. This is exactly formula (1).

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

□