

Computing the throughput of probabilistic streaming applications with replication

(Regular Submission)

A. Benoit^{1,*}, F. Dufossé¹, M. Gallet¹, B. Gaujal², Y. Robert¹

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 streaming applications. Given a streaming application whose dependence graph is a linear chain, a fully heterogeneous target platform, and a one-to-one mapping of the application onto the platform (a processor is assigned at most one application stage), how can we compute the throughput of the application, i.e., the rate at which data sets can be processed? The problem is easy when workflow 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 case when execution and communication times are deterministic, there are examples where the optimal period (i.e., the inverse of the throughput) is larger than the largest cycle-time of any resource. We model the problem as a timed Petri net to compute the optimal throughput in the general case, and we show how the throughput is impacted when execution and communication times are no longer deterministic, but follow some random variable laws. Finally, we prove that the problem of finding a one-to-one mapping with replication which maximizes the throughput is NP-complete, even with no communication costs.

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

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 [8, 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. With this definition, the system can process data sets at a rate $\rho = 1/\mathcal{P}$ (the throughput).

The workflow is executed on a fully heterogeneous platform, whose processors have different speeds, and whose interconnection links have different bandwidths. When mapping application stages onto processors, we enforce the rule that any given processor will execute at most one stage. However, the converse is not true. If the computations of a given stage are independent from one data set to another, then two consecutive computations (different data sets) for 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 [6, 14, 19]. This corresponds to the *dealable* stages of Cole [7].

Given an application and a target heterogeneous platform, the problem to determine the optimal mapping (maximizing the throughput) has been shown NP-hard in [5] when a processor can handle several application stages. We have thus investigated in [2] the problem of computing the throughput for a given mapping, which turned out to be also a complex problem. Indeed, with general mappings in which a processor handles several stages, the core problem is to order communications so as to optimize the throughput of the application. This problem turns out to be NP-complete in some cases.

In this paper, we focus on a particular class of mappings, namely *one-to-one mappings*, in which a processor is assigned at most one stage. In such a case, there are no problems of communication ordering. One-to-one mappings were also studied in [1], but for more complex applications in which stages are filtering data, and can have arbitrary dependencies (while we focus in this work on a linear chain of stages). Due to these dependencies, this study implies to order communications as well so as to maximize the throughput.

In the current work, we focus on one-to-one mappings, which are easier to analyze than general mappings. Indeed, the problem of computing the throughput of a given one-to-one mapping is then easy since the period is dictated by the critical hardware resource. However, we investigate here the use of *replication*: each stage of the application pipeline can be mapped onto several distinct processors. In this case, we showed in [4] that the problem gets surprisingly complicated, and we provided examples where the optimal period is larger than the largest cycle-time of any resource. In other words, during the execution of the system, all the resources will be idle at some points. We then showed how to use timed Petri nets to compute the throughput in the general deterministic case.

Here, we extend the previous approach to *probabilistic* streaming applications: we show how to compute (or bound) the throughput of the application when the communication and computation times are no longer deterministic, but follow some random variations. Also, we fill in this paper an important complexity gap for the more complex problem of finding a mapping which maximizes the throughput of the application: we demonstrate that the problem is NP-hard for one-to-one mappings with replication, even in the deterministic case with no communication costs.

First in Section 2, we describe formally the framework, 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 exponential laws (Section 3). We give a general method which turns out to be of exponential complexity in some cases, and provide a polynomial algo-

rithm for a simpler instance of the problem. Then in Section 4, we compare results when times follow general I.I.D. variables. Finally, we come back to the problem of finding the optimal mapping and prove its NP-completeness in Section 5. We conclude 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, and then we formally define the optimization problems that we target in Section 2.3. Before moving to the probabilistic study, we recall the results already established for the deterministic case in Section 2.4. Finally, we give a detailed presentation of the random variables that we consider to model the processor speeds and link bandwidths (Section 2.5).

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_k ($0 \leq k \leq n - 1$). Each stage T_k has a size w_k , expressed in flop, and needs an input file F_{k-1} of size δ_{k-1} , expressed in bytes. Finally, T_k produces an output file F_k of size δ_k , which is the input file of stage T_{k+1} . All these sizes are independent of the data set. Note that T_0 produces the initial data and does not receive any input file, while T_{n-1} gathers the final data and does not send any file.

The workflow is executed on a fully heterogeneous platform with p processors. The speed of processor P_u is denoted as Π_u (in flops). We assume bidirectional links $\text{link}_{u,v} : P_u \rightarrow P_v$ between any processor pair P_u and P_v , with bandwidth $b_{u,v}$ 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_u to P_v is $\frac{\delta_i}{b_{u,v}}$, while the time needed to process T_k on P_u is $\frac{w_k}{\Pi_u}$.

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 data set $i + 1$, compute the result of data set i and send the resulting data set $i - 1$ to the next processor. 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. Such mappings are usually called *one-to-one mappings*, see for instance [5]. Actually, we allow stage replication, so we 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*, using the terminology of Subhlok and Vondran [15, 16] and of the DataCutter team [6, 14, 19]. This corresponds to the *dealable* stages of Cole [7].

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 from previous results for other data sets, hence the possibility to process different

data sets in different locations. The following schema illustrates the replication of a stage T_k onto three processors:

$$\begin{array}{ccccc} & & / & T_k \text{ on } P_1: \text{ data sets } \mathbf{1, 4, 7, \dots} & \backslash \\ \dots & T_{k-1} & \text{---} & T_k \text{ on } P_2: \text{ data sets } \mathbf{2, 5, 8, \dots} & \text{---} & T_{k+1} & \dots \\ & & \backslash & T_k \text{ on } P_3: \text{ data sets } \mathbf{3, 6, 9, \dots} & / \end{array}$$

Let R_i denotes the number of processors participating to the process of T_i . If P_k participates to the work of T_i , then let R'_k be equal to 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 on the non-replicated stage. As a result, round-robin execution is enforced in all the papers referenced above, and we enforce this rule too.

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

2.3 Problem definition

The objective is to find a mapping of the application stages onto the processors, using replication techniques, so as to maximize the throughput ρ of the system. The throughput 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 $C_{\text{exec}}(k)$ be the cycle-time of processor P_k . If we enforce the **Overlap** model, then $C_{\text{exec}}(k)$ is equal to the maximum of its reception time $C_{\text{in}}(k)$, its computation time $C_{\text{comp}}(k)$, and its transmission time $C_{\text{out}}(k)$ (note that $C_{\text{in}}(0) = C_{\text{out}}(n-1) = 0$): $C_{\text{exec}}(k) = \max \{C_{\text{in}}(k), C_{\text{comp}}(k), C_{\text{out}}(k)\}$. If we enforce the **Strict** model, then $C_{\text{exec}}(k)$ is equal to the sum of the three operations: $C_{\text{exec}}(k) = C_{\text{in}}(k) + C_{\text{comp}}(k) + C_{\text{out}}(k)$.

Note that in both models, the maximum cycle-time, $\mathcal{M}_{\text{ct}} = \max_{1 \leq k \leq p} C_{\text{exec}}(k)$, is a lower bound for the period.

Besides the important problem of finding the best mapping, we focus in this paper on the following problem, which in appearance looks simpler: given the mapping of stages to processors, how can we compute the throughput ρ ? If no stage is replicated, then the throughput is simply determined by the critical resource (maximum cycle-time): $\rho = 1/\mathcal{M}_{\text{ct}}$. Again, this problem is addressed in [5] for the **Strict** model but the same result can be easily shown for the **Overlap** model. However, when stages are replicated, the previous result is no longer true, and we need to use more sophisticated techniques to compute the throughput, such as timed Petri nets. We recall the main results that we established previously [4] in the next section, while we will come back to the problem of finding the best mapping in Section 5.

2.4 Computing the throughput in the deterministic case

The problem of computing the throughput when execution and communication times are deterministic was already studied in [4]. We recall the main results for the two communication models.

- **Overlap**: the throughput can be determined in polynomial time.

- **Strict:** determining the complexity of this problem remains open. There already exists a method to compute the throughput in time $O(\text{lcm}_{1 \leq i \leq m-1}(R_i)^3)$, leading to 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.

2.5 Random variations

We consider in the following that the time to complete a stage and the time to transfer data are random variables. Moreover, each processor deals with only one stage (see Section 2.2). Thus, in the deterministic case, we can denote the n -th computation time of stage T_k on processor P_i by $c_i = w_k/s_k$. Similarly, the n -th communication time of the file F_k sent by P_i to P_j is given by $d_{i,j} = \delta_k/b_{i,j}$.

In the case of probabilistic communication and computation times, we denote by $X_i(n)$ the random variable giving the actual computation time of the n -th time stage T_k processed by P_i . Similarly, we denote by $Y_{i,j}(n)$ the random variable giving the actual communication of the n -th file transferred from P_i to P_j . Thus, we further denote the mapping of an application (T_1, \dots, T_m) on a platform (P_1, \dots, P_n) by $((X_i(n)), (Y_{i,j}(n)))$.

The probability that the computation time of T_k on P_i is larger than x is given by $\Pr(X_i(n) > x)$, while its expected value is given by $\mathbf{E}[X_i(n)]$.

This definition is general and does not imply any special constraint on the involved random variables. However, some presented results are only valid for specific classes of random variables. Below we recall the definition of these other classes of random variables.

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 $\Pr(X > t) = 1 - e^{-\lambda t}$. The value of this variable is equal to $1/\lambda$.

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 $t > 0$. 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 $t > 0$). 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. 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 always be true in the following: Processing times $\{X_i(n)\}_{n \in \mathbb{N}}$ and communication times $\{Y_{i,j}(n)\}_{n \in \mathbb{N}}$ are independent I.I.D. sequences.

3 Computing the throughput with exponential laws

In this section, we only 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

transition $\mathcal{T}_i(n)$ 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 construction principles of the timed Petri net model (Section 3.1). Then, we explain in Section 3.2 the general method which allows us to compute the throughput for both the **Overlap** and the **Strict** models. However, this method has a high complexity. In the **Overlap** case, we were able to provide a simpler method, building upon the relative simplicity of the timed Petri net (Section 3.3). Finally, we derive a polynomial algorithm for the **Overlap** case when we further assume that the communication network is homogeneous in Section 3.4.

3.1 Timed Petri net models

As in [4], we model the system formed by a given mapping of an application onto a platform by a timed Petri net. Here we briefly recall the construction principles:

- any path followed by the input data sets is fully developed into the timed Petri net,
- any computation or communication is represented by a transition, whose firing time is the same as the computation time (respectively communication time),
- dependences between two successive operations are represented by places between the transitions corresponding to these operations.

The complete timed Petri net representing Example A (Figure 1) is shown in Figure 2. Note that in all cases, the 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 $((X_i(n)), (Y_{i,j}(n)))$ formed by the mapping of an application onto a platform. Then the throughput can be computed in time $O(\exp(\text{lcm}_{1 \leq i \leq m-1}(R_i))^3)$.*

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

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.

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, giving an easier problem than the **Strict** model.

Theorem 2. *Let us consider the system $((X_i(n)), (Y_{i,j}(n)))$ formed by the mapping of an application onto a platform, following the **Overlap** communication model. Then the throughput can be computed in time $O(m \exp(\max_{1 \leq i \leq m-1}(R_i))^3)$.*

Here again, due to a lack of space, we only give the overall structure of the proof, while we refer to the appendix for a detailed proof:

1. split the timed Petri net into several columns C_i ;
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} ;

5. transform \mathcal{P} into a Markov chain C ;
6. determine a stationary measure of C ;
7. compute the throughput of \mathcal{P} in isolation (called inner throughput in the following);
8. combine the 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 [4].

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 C_i , denoted λ_i , then the inner throughput of each strongly connected component 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 $((X_i(n)), (Y_{i,j}(n)))$ formed by the mapping of an application onto a platform, following the **Overlap** communication model with a homogeneous communication network. The inner throughput of a processor component C_k is $\rho_k = \lambda_k$.*

The inner throughput of a communication strongly connected component C_i is equal to $\rho_i = \frac{\lambda_i}{v_i + u_i - 1}$. The global throughput can be computed in polynomial time from all inner throughputs and is equal to:

$$\rho = \sum_{C_i \in \text{column } m} \min_{C_j \prec C_i} \rho_j, \quad (1)$$

where $C_j \prec C_i$ means that there exists a path from component C_j to component C_i .

Before we prove this main result, let us give several comments. First, the inner throughput of one strongly connected component C_i under exponential communication times given by the very simple formula $\rho_i = \frac{\lambda_i}{v_i + u_i - 1}$ is to be compared with the throughput in the deterministic case where all communication times are deterministic ($1/\lambda_i$), equal to $\rho_i = \frac{\lambda_i}{\max(v_i, u_i)}$. 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 [3].

Proof. Platforms with **Overlap** model and 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 C .

If C corresponds to a processor, the formula given previously applies and its inner throughput is $\rho_C = \lambda_C$.

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 C . 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 any of the ready transitions. On Figure 8, there are 4 fireable transitions, shown as 4 top-left corners on the bold line. Moreover, this reasoning can be inverted: any bottom-right corner can be inverted, giving a possible previous state leading to the current state.

Since we have as many top-left corners as bottom-right ones, any state of C has the same number of incoming states as the outgoing ones. Moreover, because the communication network is homogeneous, all transitions have the same firing rate. These two conditions imply that the Markov chain C admits the

uniform measure as invariant measure [11]: if N is the number of states of C , then its invariant measure is $(\frac{1}{N}, \dots, \frac{1}{N})$.

Last, let us compute the number of states of C allowing a given transition to be fired. Due to the round-robin distribution of communications, all transitions have the same firing rate and we can only consider the top-right transition T . The number of states such that T is fireable is exactly the number $M(u, v)$ of possible paths starting from this top-right corner. $M(u, v)$ is computed in the same way as $N(u, v)$ (see proof of Theorem 2):

$$M(u, v) = \sum_{i=0}^{u-2} \sum_{j=0}^{v-2} \alpha_{i,j} = \binom{u+v-2}{u-1} = \frac{N(u, v)}{v+u-1}.$$

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

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 to the last.

Now the global throughput 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 m . 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 makes the computation of this formula polynomial in the number of tasks and processors.

□


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, even in polynomial time for the homogeneous **Overlap** case.

Previous works summarized in Section 2.4 have presented methods to compute the throughput in the case of deterministic communication times and processing times.

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 [12]. 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.

We provide a few numerical results in Section 4.2 to illustrate the behavior of several variables...

 **completer quand on aura la section...**

4.1 Theoretical comparison

Definition 1. Let $\{X(n)\}_{n \in \mathbb{N}}$ and $\{Y(n)\}_{n \in \mathbb{N}}$ be two real random variable sequences.

- X is smaller than Y for the strong order (denoted $X \leq_{\text{st}} Y$) if for all increasing function f , $\mathbf{E}[f(X(1), X(2), \dots)] \leq \mathbf{E}[f(Y(1), Y(2), \dots)]$.
- X is smaller than Y for the increasing convex order (denoted $X \leq_{\text{icx}} Y$) if for all increasing convex function g , $\mathbf{E}[g(X(1), X(2), \dots)] \leq \mathbf{E}[g(Y(1), Y(2), \dots)]$.

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

Theorem 4. *Let us consider two systems $((X_i^{(1)}(n)), (Y_{i,j}^{(1)}(n)))$ and $((X_i^{(2)}(n)), (Y_{i,j}^{(2)}(n)))$.*

If $\forall i \leq m, X_i^{(1)}(n) \leq_{st} X_i^{(2)}(n)$ and $\forall i, j \leq m-1, Y_{i,j}^{(1)}(n) \leq_{st} Y_{i,j}^{(2)}(n)$, 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 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: $D_k^b(n) = D^b(n-1) \otimes A^b(n)$ 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 T_i and T_j 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 $T_k^b(n) = X_i^{(b)}(n)$ or $T_k^b(n) = Y_{i,j}^{(b)}(n)$. 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 $D^1(n) \leq_{st} D^2(n)$.

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. \square

Theorem 5. *Let us consider two systems with I.I.D. communication and processing times*

$((X_i^{(1)}(n)), (Y_{i,j}^{(1)}(n)))$ and $((X_i^{(2)}(n)), (Y_{i,j}^{(2)}(n)))$. If we have for all n ,

$\forall i \leq m, X_i^{(1)}(n) \leq_{icx} X_i^{(2)}(n)$ and $\forall i, j \leq m-1, Y_{i,j}^{(1)}(n) \leq_{icx} Y_{i,j}^{(2)}(n)$, then we have

$$\rho^{(1)} \geq \rho^{(2)}.$$

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. \square

Theorem 6. *Let us consider any system $((X_i^{(1)}(n)), (Y_{i,j}^{(1)}(n)))$, such that $X_i^{(1)}(n)$ and $Y_{i,j}^{(1)}(n)$ are N.B.U.E.. Let us also consider two new systems $((X_i^{(2)}(n)), (Y_{i,j}^{(2)}(n)))$ and $((X_i^{(3)}(n)), (Y_{i,j}^{(3)}(n)))$ such that:*

- $\forall i \leq m, X_i^{(2)}(n)$ has an exponential distribution, and $\mathbf{E}[X_i^{(1)}(n)] = \mathbf{E}[X_i^{(2)}(n)]$,
- $\forall i, j \leq m-1, Y_{i,j}^{(2)}(n)$ has an exponential distribution, and $\mathbf{E}[Y_{i,j}^{(1)}(n)] = \mathbf{E}[Y_{i,j}^{(2)}(n)]$,
- $\forall i \leq m, X_i^{(3)}(n)$ is deterministic and for all $n, X_i^{(3)}(n) = \mathbf{E}[X_i^{(2)}(n)]$,
- $\forall i, j \leq m-1, Y_{i,j}^{(3)}(n)$ is deterministic and for all $n, Y_{i,j}^{(3)}(n) = \mathbf{E}[Y_{i,j}^{(2)}(n)]$.

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 [10], 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 $\rho^{(3)} \geq \rho^{(1)} \geq \rho^{(2)}$. \square

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

In particular, this implies that in an **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).

4.2 Numerical experiments

Author: Matthieu

5 Finding the optimal mapping

In this section, we come back to our original problem, consisting in finding a one-to-one mapping which maximizes the throughput when allowing stage replication. The problem was shown to be NP-hard in [5] when no stage can be replicated (thereby enforcing a one-to-one mapping of stages to processors). The proof of [5] was given for the **Strict** model but can be easily extended to the **Overlap** model.

However, with homogeneous communications, the problem was of polynomial complexity with no replication (see [5]). Our new contribution is to establish the NP-completeness of this problem when considering replicated stages, even with no communication costs. This result is established for the deterministic case, and thus the complexity holds for the probabilistic study.

Theorem 7. *In the deterministic case, the problem of finding the one-to-one 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 , is 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 [9]. Let \mathcal{I}_1 be an instance of 3-PARTITION: given a set $A = \{a_1, \dots, a_{3m}\}$ 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, \dots, 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 could encode \mathcal{I}_1 in unary. Moreover, 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, \dots, 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 same stage have the same speed, 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 $\forall j, \sum_{a_k \in A_j} a_k \times m! \leq j \times B \frac{m!}{j}$, that means $\forall j, \sum_{a_k \in A_j} a_k \leq B$, and since we have $\sum_{1 \leq i \leq 3m} a_i = mB$, it means that $\forall j, \sum_{a_k \in A_j} a_k = B$. Therefore, A_1, \dots, 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 of probabilistic streaming applications. Indeed, the problem turns out difficult when stage replication is performed in a round-robin fashion. In previous work [4], we provided methods to compute the throughput in a deterministic case, using timed Petri nets. We extended these results in a case where computation and communication times follow random variable laws. In the case of exponential laws, we identified cases in which we were able to compute the throughput explicitly. In the general case, we were able to give bounds to the throughput when random variables are I.I.D. N.B.U.E.. Moreover, the lower and upper bounds are obtained by the deterministic and exponential cases respectively, and we can compute both bounds in polynomial time under the **Overlap** model with a homogeneous communication network.

Computing the throughput given the mapping is an important problem but one may further want to decide of a mapping which provides the optimal throughput. We proved that this problem is NP-complete, even in the simpler deterministic case with no communication costs.

Now that we have new methods to evaluate the throughput of a given mapping even in a probabilistic setting, we plan in future work to design 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 important step in the field of scheduling streaming applications on a dynamic platform subject to variations.

Acknowledgment

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] K. Agrawal, A. Benoit, F. Dufossé, and Y. Robert. Mapping filtering streaming applications with communication costs. In *21st ACM Symposium on Parallelism in Algorithms and Architectures SPAA 2009*. ACM Press, 2009. Also available as LIP Research Report 2009-06 at graal.ens-lyon.fr/~abenoit.
- [2] K. Agrawal, A. Benoit, L. Magnan, and Y. Robert. Scheduling algorithms for linear workflow optimization. In *IPDPS'2010, the 24th IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society Press, 2010. To appear, also available as LIP Research Report 2009-22 at graal.ens-lyon.fr/~abenoit.
- [3] F. Baccelli and D. Hong. Analyticity of iterates of random non-expansive maps. Research Report 3558, INRIA, Sophia-Antipolis, 1998.
- [4] 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.
- [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] M. Cole. Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. *Parallel Computing*, 30(3):389–406, 2004.
- [8] DataCutter Project: Middleware for Filtering Large Archival Scientific Datasets in a Grid Environment. <http://www.cs.umd.edu/projects/hpsl/ResearchAreas/DataCutter.htm>.
- [9] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [10] B. Gaujal and J.-M. Vincent. *Introduction to Scheduling*, chapter Comparisons of stochastic task-resource systems. CC Press, 2009.
- [11] O. Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.
- [12] J. Kamburowski. Bounding the distribution of project duration in pert networks. *Operation Research Letters*, 12:17–22, 1992.
- [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.

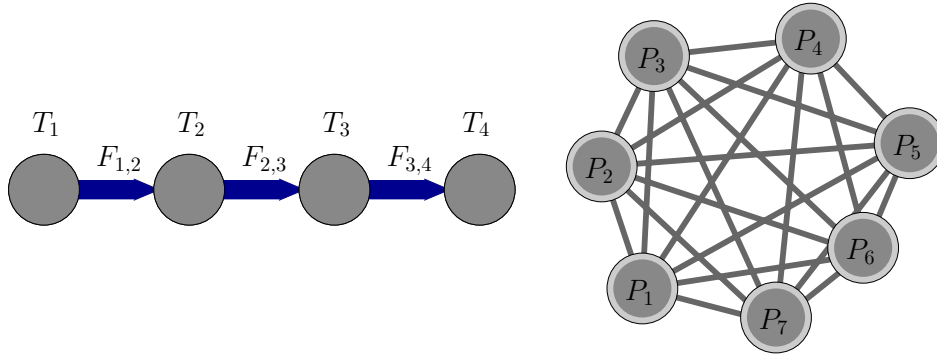


Figure 1: Example A: Four-stage pipeline and seven-processor computing platform

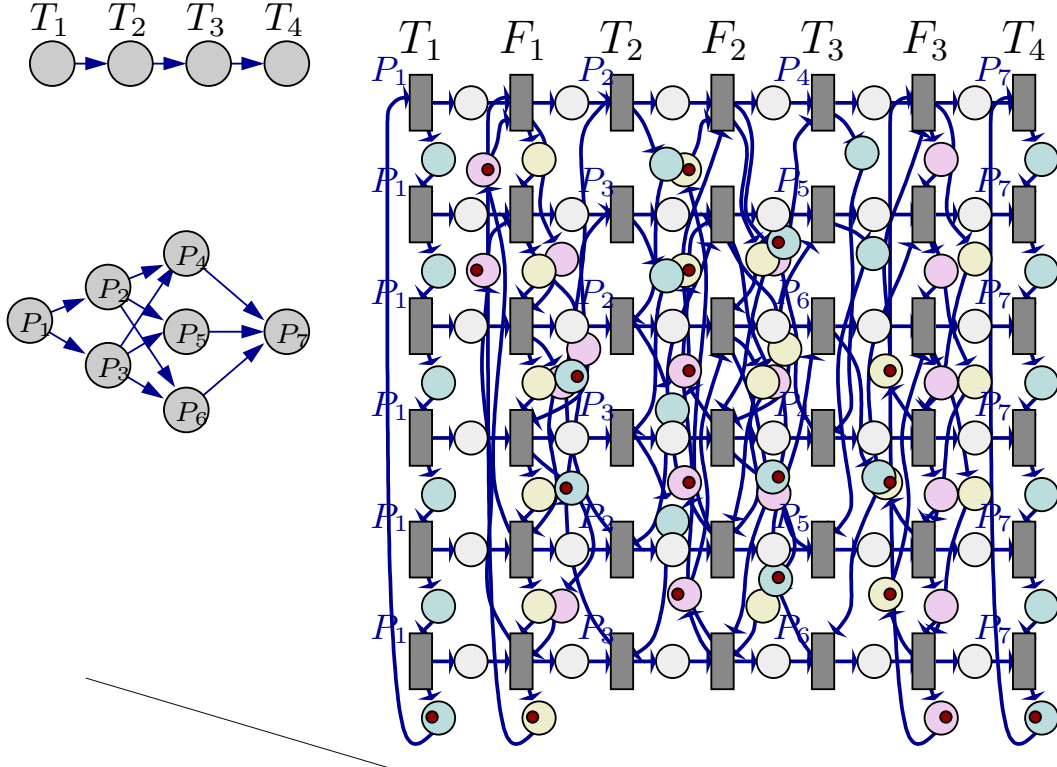


Figure 2: Timed Petri net representing Example A.

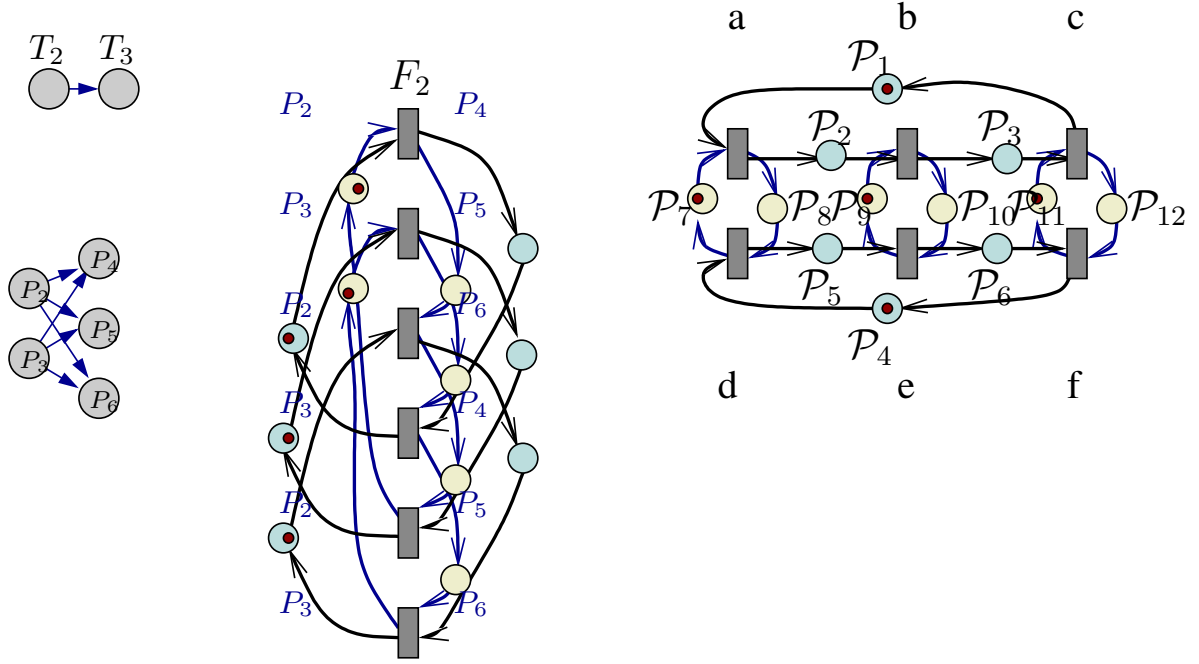


Figure 3: Example A: Part of the timed Petri net corresponding to communication F_2 .

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 [4] and we do not detail it here. This step is done in time $O(Rm)$, 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, \dots) . To each possible marking \mathcal{M}_i of the timed Petri net, we associate a state x_i . There are $(2m + 3(m - 1))R$ places, and each place contains either zero or one token. Thus, there are at most $2^{(2m+3(m-1))R}$ 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 3.

On Example A, places are named $(P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}, P_{12})$, 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

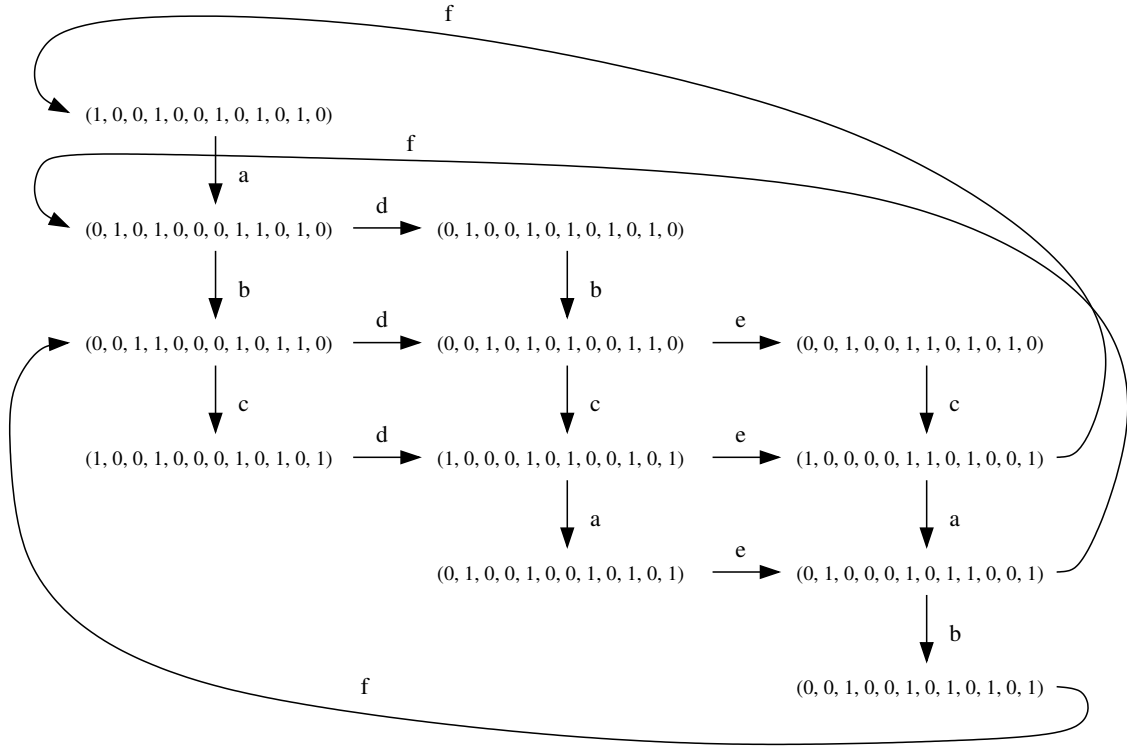


Figure 4: List of all possible states of the Markov chain corresponding to the reduced timed Petri net of Example A.

are labeled with the names of the transitions. The complete list of possible states and the corresponding transitions are given in Figure 4.

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_m 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 [11].

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 several columns C_i ;
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} ;
5. transform \mathcal{P} into a Markov chain C ;
6. determine a stationary measure of C ;
7. compute the throughput of \mathcal{P} in isolation (called inner throughput in the following);
8. combine the throughputs of all components to get the global throughput of the system.

To decrease the overall complexity, we use the same idea as in [4]: 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; (ii) the column corresponds to communications between two sets of processors.

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 $X_i(n)$ being exponential, this is equal to the rate λ_i of X_i .

On the contrary, case (ii) is more complex and requires a more detailed study. Let us consider the i -th communication: 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 is made of $c = \frac{R}{\text{lcm}(R_i, R_{i+1})}$ copies of a pattern of size $u \times v$. Since these components are independent, we can compute the throughput of each of them independently. In the case of Example B presented in Figure 5, 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} of size $u \times v = 9 \times 7$.

Each pattern is a timed Petri net \mathcal{P} with a very regular structure, which can be represented as a rectangle of size (u, v) , as shown in Figure 5. 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 $O(t^3)$ [11]. Thus, we need to determine the number of states of the transformation of \mathcal{P} into a Markov chain. Let C be this Markov chain.

The number of states of C is by definition the number of possible markings, and we can directly determine it. A valid marking of \mathcal{P} of Figure 5 is represented in Figure 6. 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, \dots, 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, \dots, 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 6. 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 C . 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 7). If i and j are given, then there are

$\alpha_{i,j} = \binom{i+j}{i}$ possible paths from $(i, 0)$ to $(0, j)$, where $\binom{n}{k}$ 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 $N(u, v)$ of valid markings can be easily determined:

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

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

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

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

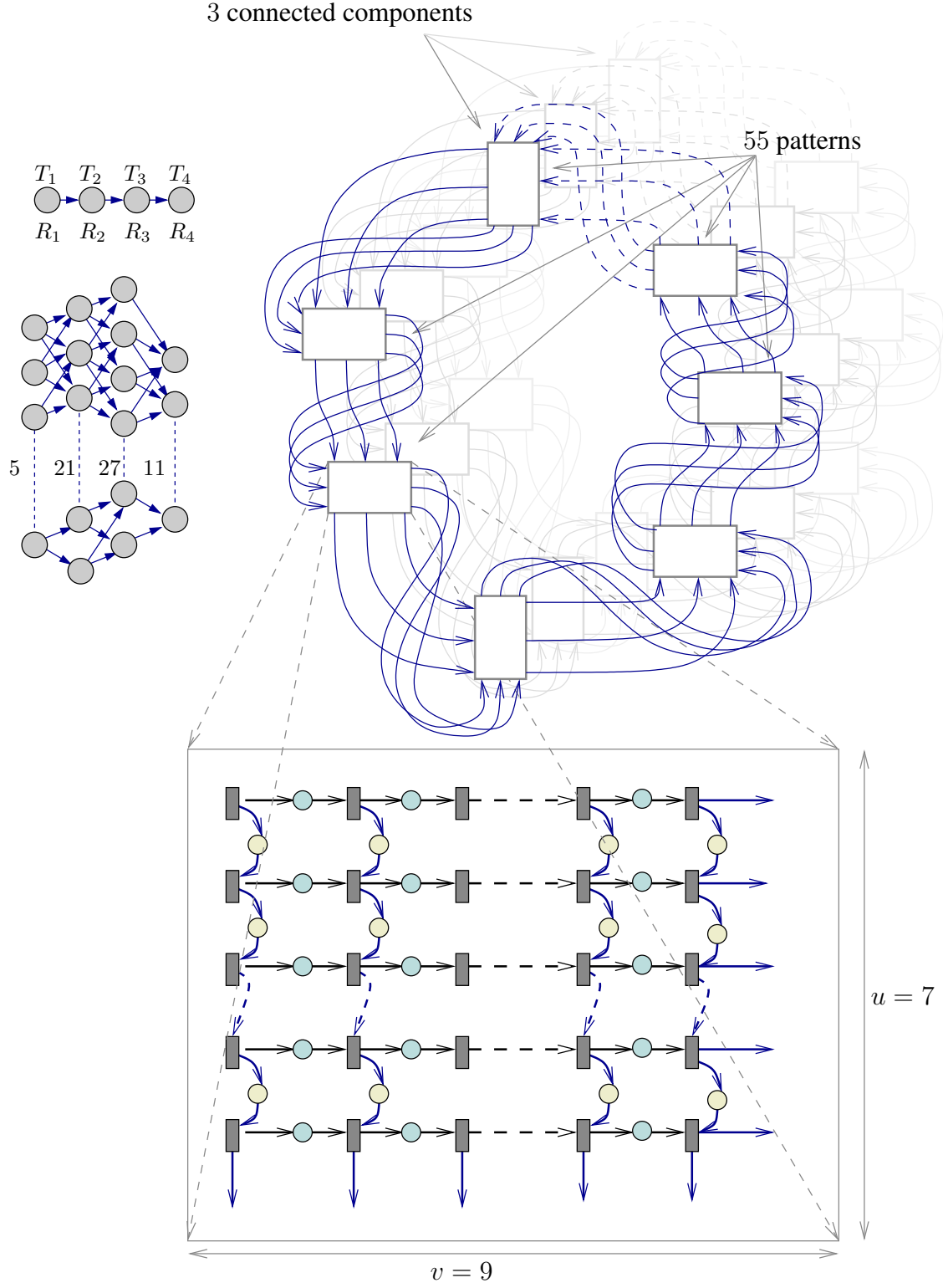


Figure 5: Example B, with stages replicated on 5, 21, 27 and 11 processors, and structure of the timed Petri net corresponding to the second communication.

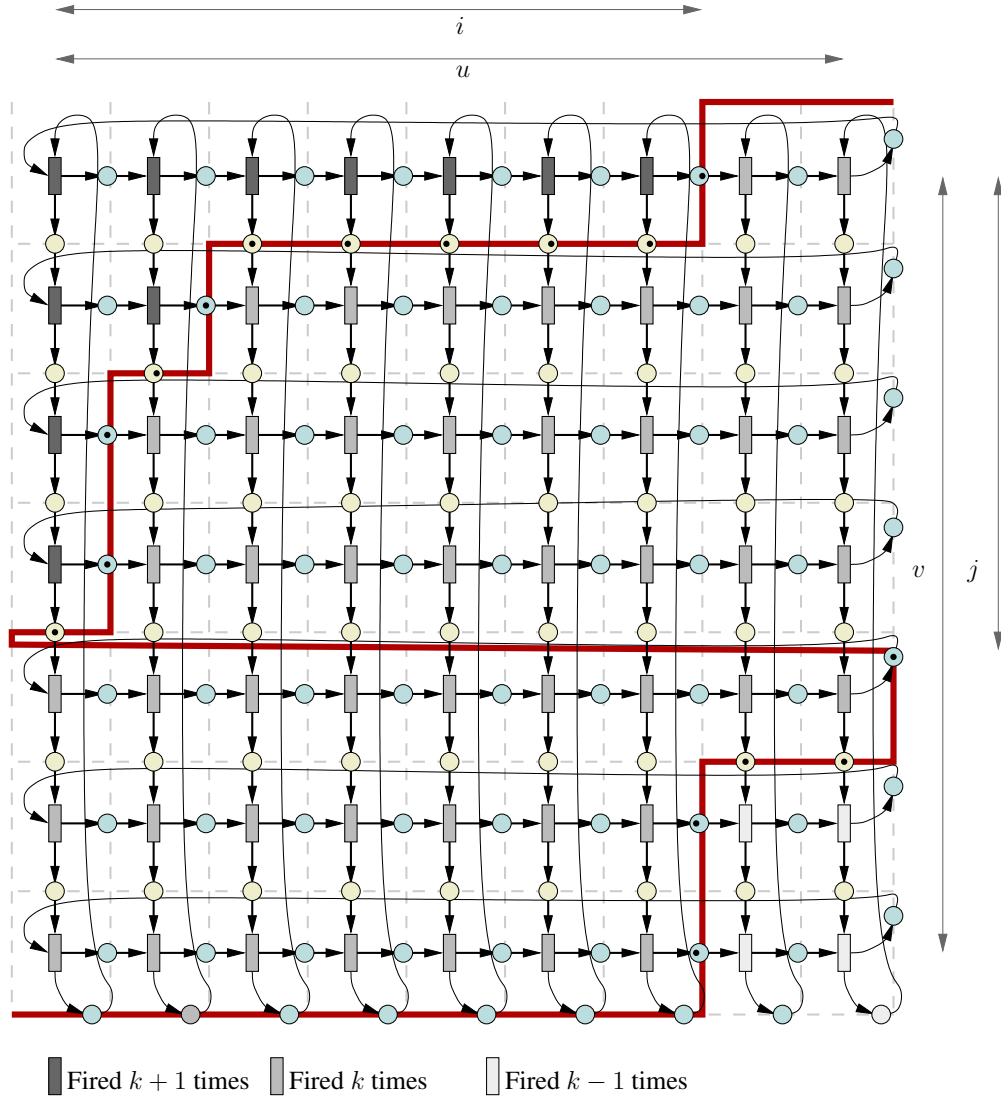


Figure 6: Valid marking of \mathcal{P} , the reduced timed Petri net of the second communication of Example B.

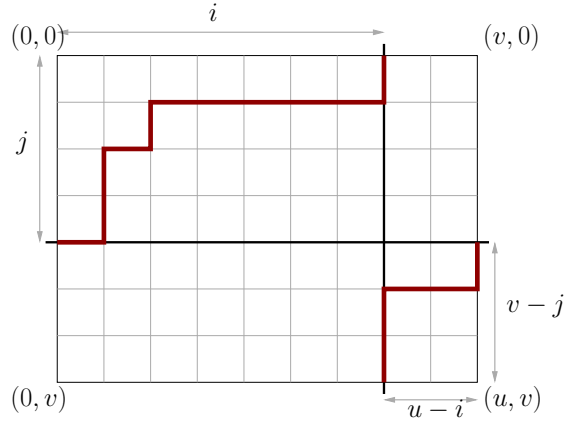


Figure 7: Representation with Young diagrams of a valid marking.

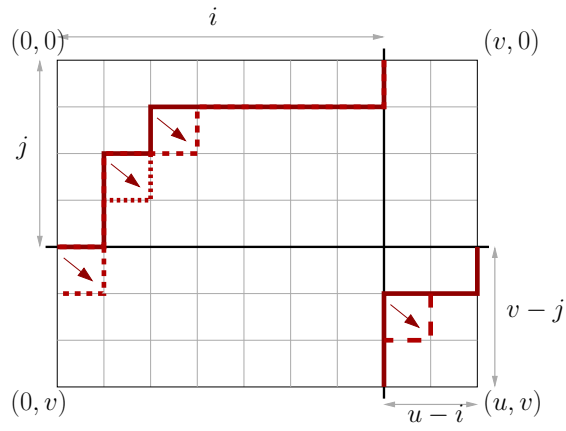


Figure 8: Reachable states from a given position.