



HAL
open science

Optimizing latency and reliability of pipeline workflow applications

Anne Benoit, Veronika Rehn-Sonigo, Yves Robert

► **To cite this version:**

Anne Benoit, Veronika Rehn-Sonigo, Yves Robert. Optimizing latency and reliability of pipeline workflow applications. [Research Report] LIp RR-2007-43, Laboratoire de l'informatique du parallélisme. 2007, 2+11p. hal-02102604

HAL Id: hal-02102604

<https://hal-lara.archives-ouvertes.fr/hal-02102604>

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

*Optimizing Latency and Reliability of
Pipeline Workflow Applications*

Anne Benoit ,
Veronika Rehn-Sonigo ,
Yves Robert

November 2007

Research Report N° RR2007-43

École Normale Supérieure de Lyon

46 Allée d'Italie, 69364 Lyon Cedex 07, France

Téléphone : +33(0)4.72.72.80.37

Télécopieur : +33(0)4.72.72.80.80

Adresse électronique : lip@ens-lyon.fr



Optimizing Latency and Reliability of Pipeline Workflow Applications

Anne Benoit , Veronika Rehn-Sonigo , Yves Robert

November 2007

Abstract

Mapping applications onto heterogeneous platforms is a difficult challenge, even for simple application patterns such as pipeline graphs. The problem is even more complex when processors are subject to failure during the execution of the application. In this paper, we study the complexity of a bi-criteria mapping which aims at optimizing the latency (*i.e.*, the response time) and the reliability (*i.e.*, the probability that the computation will be successful) of the application. Latency is minimized by using faster processors, while reliability is increased by replicating computations on a set of processors. However, replication increases latency (additional communications, slower processors). The application fails to be executed only if all the processors fail during execution. While simple polynomial algorithms can be found for fully homogeneous platforms, the problem becomes NP-hard when tackling heterogeneous platforms. This is yet another illustration of the additional complexity added by heterogeneity.

Keywords: Heterogeneity, scheduling, complexity results, reliability, response time.

Résumé

L'ordonnancement et l'allocation des applications sur plates-formes hétérogènes sont des problèmes cruciaux, même pour des applications simples comme des graphes en pipeline. Le problème devient même encore plus complexe quand les processeurs peuvent tomber en panne pendant l'exécution de l'application. Dans cet article, nous étudions la complexité d'une allocation bi-critère qui vise à optimiser la latence (*i.e.*, le temps de réponse) et la fiabilité (*i.e.*, la probabilité que le calcul réussisse) de l'application. La latence est minimisée en utilisant des processeurs rapides, tandis que la fiabilité est augmentée en répliquant les calculs sur un ensemble de processeurs. Toutefois, la réplication augmente la latence (communications additionnelles et processeurs moins rapides). L'application échoue à être exécutée seulement si tout les processeurs échouent pendant l'exécution. Des algorithmes simples en temps polynomial peuvent être trouvés pour plates-formes complètement homogènes, tandis que le problème devient NP-dur quand on s'attaque aux plates-formes hétérogènes. C'est encore une autre illustration de la complexité additionnelle due à l'hétérogénéité.

Mots-clés: Hétérogénéité, ordonnancement, résultats de complexité, fiabilité, temps de réponse.

1 Introduction

Mapping applications onto parallel platforms is a difficult challenge. Several scheduling and load-balancing techniques have been developed for homogeneous architectures (see [13] for a survey) but the advent of heterogeneous clusters has rendered the mapping problem even more difficult. Moreover, in a distributed computing architecture, some processors may suddenly become unavailable and we are confronted to the problem of failure [1, 2]. In this context of dynamic heterogeneous platforms with failures, a structured programming approach rules out many of the problems which the low-level parallel application developer is usually confronted to, such as deadlocks or process starvation. In this paper, we consider application workflows that can be expressed as pipeline graphs. A series of data sets (tasks) enter the input stage and progress from stage to stage until the final result is computed. Each stage has its own communication and computation requirements: it reads an input file from the previous stage, processes the data and outputs a result to the next stage. For each data set, initial data is input to the first stage, and final results are output from the last stage.

Each processor has a failure probability, which expresses the chance that the processor fails during execution. Key metrics for a given workflow are the latency and the failure probability. The latency is the time elapsed between the beginning and the end of the execution of a given data set, hence it measures the response time of the system to process the data set entirely. Intuitively, we minimize the latency by assigning all stages to the fastest processor, but this may lead to an unreliable execution of the application. Therefore, we need to find trade-offs between two antagonistic objectives, namely latency and failure probability. Informally, the application will be reliable for a given mapping if the corresponding global failure probability is small. In this paper, we focus on bi-criteria approaches, i.e., minimizing the latency under failure probability constraints, or the converse. Indeed, such bi-criteria approaches seem more natural than the minimization of a linear combination of both criteria. Users may have latency constraints or reliability constraints, but it makes little sense for them to minimize the sum of the latency and of the failure probability.

In this paper, we focus on pipeline skeletons and thus we enforce the rule that a given stage is mapped onto a single processor. In other words, a processor that is assigned a stage will execute the operations required by this stage (input, computation and output) for all the tasks fed into the pipeline. However, in order to improve reliability, we can replicate the computations for a given stage on several processors, i.e., a set of processors performs identical computations on every data set. Thus, in case of failure, we can take the result from a processor which is still working. The optimization problem can be stated informally as follows: which stage to assign to which (set of) processors? We require the mapping to be interval-based, i.e., a set of processors is assigned an interval of consecutive stages. The main objective of this paper is to assess the complexity of this bi-criteria mapping problem. The rest of the paper is organized as follows. Section 2 is devoted to the presentation of the target optimization problems. Next in Section 3 we proceed to the complexity results. Finally, we briefly review related work and state some concluding remarks in Section 4.

2 Framework and optimization problems

The application is expressed as a pipeline graph of n stages \mathcal{S}_k , $1 \leq k \leq n$, as illustrated on Figure 1. Consecutive data sets are fed into the pipeline and processed from stage to stage,

until they exit the pipeline after the last stage. Each stage executes a task. More precisely, the k -th stage \mathcal{S}_k receives an input from the previous stage, of size δ_{k-1} , performs a number of w_k computations, and outputs data of size δ_k to the next stage. This operation corresponds to the k -th task and is repeated periodically on each data set. The first stage \mathcal{S}_1 receives an input of size δ_0 from the outside world, while the last stage \mathcal{S}_n returns the result, of size δ_n , to the outside world.

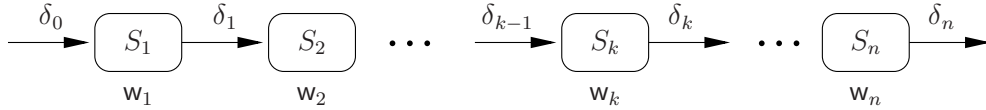


Figure 1: The application pipeline.

We target a platform with m processors P_u , $1 \leq u \leq m$, fully interconnected as a (virtual) clique. We associate to each processor a failure probability $0 \leq \text{fp}_u \leq 1$, $1 \leq u \leq m$, which is the probability that the processor breaks down during the execution of the application. There is a bidirectional link $\text{link}_{u,v} : P_u \rightarrow P_v$ between any processor pair P_u and P_v , of bandwidth $\mathbf{b}_{u,v}$. The speed of processor P_u is denoted as \mathbf{s}_u , and it takes X/\mathbf{s}_u time-units for P_u to execute X floating point operations. We also enforce a linear cost model for communications, hence it takes $X/\mathbf{b}_{u,v}$ time-units to send (or receive) a message of size X from P_u to P_v . Communication contention is taken care of by enforcing the *one-port* model [5, 6]. In this model, a given processor can be involved in a single communication at any time-step, either a send or a receive. However, independent communications between distinct processor pairs can take place simultaneously. The one-port model seems to fit the performance of some current MPI implementations, which serialize asynchronous MPI sends as soon as message sizes exceed a few megabytes [12].

We consider three types of platforms: *Fully Homogeneous* platforms have identical processors ($\mathbf{s}_u = \mathbf{s}$ for $1 \leq u \leq m$) and interconnection links ($\mathbf{b}_{u,v} = \mathbf{b}$ for $1 \leq u, v \leq m$); *Communication Homogeneous* platforms, with identical links but different speed processors, introduce a first degree of heterogeneity; *Fully Heterogeneous* platforms constitute the most difficult instance, with different speed processors and different capacity links.

The general mapping problem consists in assigning application stages to platform processors. For simplicity, we could assume that each stage \mathcal{S}_i of the application pipeline is mapped onto a distinct processor (which is possible only if $n \leq m$). However, such one-to-one mappings may be unduly restrictive, and a natural extension is to search for interval mappings, i.e., allocation functions where each participating processor is assigned an interval of consecutive stages. Intuitively, assigning several consecutive tasks to the same processors will increase their computational load, but may well dramatically decrease communication requirements. Interval mappings constitute a natural and useful generalization of one-to-one mappings (not to speak of situations where $m < n$, where interval mappings are mandatory), and such mappings have been studied by Subhlock et al. [14, 15]. Formally, we search for a partition of $[1..n]$ into $p \leq m$ intervals $I_j = [d_j, e_j]$ such that $d_j \leq e_j$ for $1 \leq j \leq p$, $d_1 = 1$, $d_{j+1} = e_j + 1$ for $1 \leq j \leq p - 1$ and $e_p = n$.

The function $\text{alloc}(j)$ returns the indices of the processors on which interval I_j is mapped. There are $k_j = |\text{alloc}(j)|$ processors executing I_j , and obviously $k_j \geq 1$. Increasing k_j increases the reliability of the execution of interval I_j . The optimization problem is to determine the best mapping, over all possible partitions into intervals, and over all processor assignments.

The objective can be to minimize either the latency or the failure probability, or a combination: given a threshold latency, what is the minimum failure probability that can be achieved? vs. given a threshold failure probability, what is the minimum latency that can be achieved?

The failure probability can be computed given the number p of intervals and the set of processors assigned to each interval: $\mathcal{FP} = 1 - \prod_{1 \leq j \leq p} (1 - \prod_{u \in \text{alloc}(j)} \text{fp}_u)$.

We assume that $\text{alloc}(0) = \{\text{in}\}$ and $\text{alloc}(m+1) = \{\text{out}\}$, where P_{in} is a special processor holding the initial data, and P_{out} is receiving the results. Dealing with *Fully Homogeneous* and *Communication Homogeneous* platforms, the latency is obtained as

$$T_{\text{latency}} = \sum_{1 \leq j \leq p} \left\{ k_j \times \frac{\delta_{d_j-1}}{b} + \frac{\sum_{i=d_j}^{e_j} w_i}{\min_{u \in \text{alloc}(j)} (s_u)} \right\} + \frac{\delta_n}{b}. \quad (1)$$

In equation (1), we consider the longest path required to compute a given data set. The worst case is when the first processors involved in the replication fail during execution. A communication to interval j must then be paid k_j times since these are serialized (one-port model). For computations, we consider the total computation time required by the slowest processor assigned to the interval. For the final output, only one communication is required, hence the δ_n/b . Note that in order to achieve this latency, we need a standard consensus protocol to determine which of the surviving processors performs the outgoing communications [16].

A similar mechanism is used for *Fully Heterogeneous* platforms:

$$T_{\text{latency}} = \sum_{u \in \text{alloc}(1)} \frac{\delta_0}{b_{\text{in},u}} + \sum_{1 \leq j \leq p} \max_{u \in \text{alloc}(j)} \left\{ \frac{\sum_{i=d_j}^{e_j} w_i}{s_u} + \sum_{v \in \text{alloc}(j+1)} \frac{\delta_{e_j}}{b_{u,v}} \right\} \quad (2)$$

3 Complexity results

In this section, we expose the complexity results for both mono-criterion and bi-criteria problems.

3.1 Mono-criterion problems

Theorem 1. *Minimizing the failure probability can be done in polynomial time.*

This can be seen easily from the formula computing the global failure probability: the minimum is reached by replicating the whole pipeline as a single interval on all processors. This is true for all platform types.

Theorem 2. *Minimizing the latency can be done in polynomial time.*

Proof. (**Theorem 2**)

We consider *Fully Heterogeneous* platforms and we want to minimize the latency. Replication can only decrease latency so we do not consider any replication. However, we need to find the best partition into intervals.

Let us consider a directed graph with $n + m + 2$ vertices, and $(n - 1)m^2 + 2m$ edges, as illustrated in Figure 2. $V_{i,u}$ corresponds to the mapping of stage \mathcal{S}_i onto processor P_u . $V_{0,\text{in}}$ and $V_{(n+1),\text{out}}$ represent the initial and final processors, and data must flow from $V_{0,\text{in}}$ to $V_{(n+1),\text{out}}$. Edges represent the flow of data from one stage to another, thus we have m^2

edges for $i = 0..n$, connecting vertex $V_{i,u}$ to $V_{i+1,v}$ for $u, v = 1..m$ (except for the first and last stages where there are only m edges).

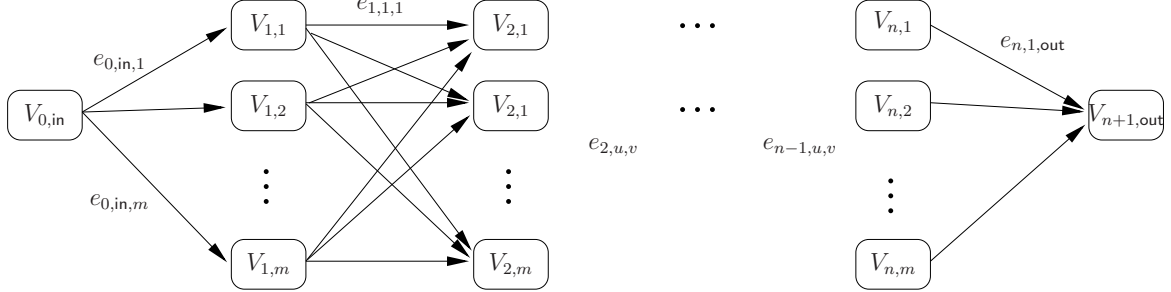


Figure 2: Minimizing the latency.

Thus, a mapping can be represented by a path from $V_{0,in}$ to $V_{(n+1),out}$: if $V_{i,u}$ is in the path then stage \mathcal{S}_i is mapped onto P_u . We assign weights to the edges to ensure that the weight of a path is the latency of the corresponding mapping. Computation cost of stage \mathcal{S}_i on P_u is added on the m edges exiting $V_{i,u}$, and thus $e_{i,u,v} = \frac{w_i}{s_u}$. Communication costs are added on all edges: $e_{i,u,v} + = \frac{\delta_i}{b_{u,v}}$ if $P_u \neq P_v$. Edges $e_{i,u,u}$ correspond to intra-interval communications, and thus there is no communication cost to pay.

The mapping which realizes the minimum latency can be obtained by finding a shortest path in this graph going from $V_{0,in}$ to $V_{(n+1),out}$. The graph has polynomial size and the shortest path can be computed in polynomial time [7], thus we have the result in polynomial time, which concludes the proof. \square

3.2 Bi-criteria problems on *Fully Homogeneous* platforms

For *Fully Homogeneous* platforms, we consider that all failure probabilities are identical, since the platform is made of identical processors. However, results can easily be extended for different failure probabilities. We prove that the optimal solution for a bi-criteria mapping on such platforms always consists in mapping the whole pipeline as a single interval. Otherwise, both latency and failure probability would be increased.

We start with a preliminary lemma which proves that the optimal solution consists of a single interval for *Fully Homogeneous* platforms, and for *Communication Homogeneous* platforms with identical failure probabilities.

Lemma 1. *On Fully Homogeneous and Communication Homogeneous-Failure Homogeneous platforms, there is a mapping of the pipeline as a single interval which minimizes the failure probability under a fixed latency threshold.*

Proof. (**Lemma 1**)

If the stages are split into m intervals, the failure probability is expressed as

$$1 - \prod_{1 \leq j \leq m} (1 - \prod_{u \in \text{alloc}(j)} \text{fp}_u)$$

Let us start with the *Fully Homogeneous* case, and with *Failure Heterogeneous* for a most general setting. We can transform the solution into a new one using a single interval, which

improves both latency and failure probability. Let k_0 be the number of times that the first interval is replicated in the original solution. Then a solution which replicates the whole interval on the k_0 most reliable processors realizes: (i) a latency which is smaller since we remove the communications between intervals; (ii) a smaller failure probability since for the new solution $(1 - \prod_{u \in \text{alloc}(1)} \text{fp}_u)$ is greater than the same expression in the original solution (the most reliable processors are used in the new one), and moreover the old solution even decreases this value by multiplying it by other terms smaller than 1. Thus the new solution is better for both criteria.

In the case with *Communication Homogeneous* and *Failure Homogeneous*, we use a similar reasoning to transform the solution. We select the interval with the fewest number of processors, denoted k . In the failure probability expression, there is a term in $(1 - \text{fp}^k)$, and thus the global failure probability is greater than $1 - (1 - \text{fp}^k)$ which is obtained by replicating the whole interval onto k processors. Since we do not want to increase the latency, we use the fastest k processors, and it is easy to check that this scheme cannot increase latency ($k \leq k_0$ and the slowest processor is not slower than the slowest processor of any intervals of the initial solution). Thus the new solution is better for both criteria, which ends the proof.

We point out that Lemma 1 cannot be extended to *Communication Homogeneous* and *Failure Heterogeneous* : instead, we can build counter examples in which this property is not true. □

Theorem 3. *On Fully Homogeneous platforms, the solution to the bi-criteria problem can be found in polynomial time using Algorithm 1 or Algorithm 2.*

Informally, the algorithm finds the maximum number of processors that can be used in the replication set, and the whole interval is mapped on this set of processors. With different failure probabilities, the more reliable processors are used.

begin

Find k maximum, such that

$$k \times \frac{\delta_0}{\mathbf{b}} + \frac{\sum_{1 \leq j \leq n} w_j}{\mathbf{s}} + \frac{\delta_n}{\mathbf{b}} \leq \mathcal{L}$$

Replicate the whole pipeline as a single interval onto the k (most reliable) processors;

end

Algorithm 1: *Fully Homogeneous* platforms: Minimizing \mathcal{FP} for a fixed \mathcal{L}

Proof. (**Theorem 3**)

The proof of this theorem is based on Lemma 1. We prove it in the general setting of heterogeneous failure probabilities. An optimal solution can be obtained by mapping the pipeline as a single interval, thus we need to decide the set of processors alloc used for replication. $|\text{alloc}|$ is the number of processors used.

The first problem can be formally expressed as follows:

$$\begin{aligned} & \text{MINIMIZE } 1 - (1 - \prod_{u \in \text{alloc}} \text{fp}_u), \\ & \text{UNDER THE CONSTRAINT} \end{aligned} \quad (3)$$

$$|\text{alloc}| \frac{\delta_0}{b} + \frac{\sum_{1 \leq i \leq n} w_i}{s} + \frac{\delta_n}{b} \leq \mathcal{L}$$

This leads to maximize $\prod_{u \in \text{alloc}} \text{fp}_u$, and the constraint on the latency determines the maximum number k of processors which can be used:

$$k = \left\lfloor \frac{b}{\delta_0} \left(\mathcal{L} - \frac{\delta_n}{b} - \frac{\sum_{1 \leq i \leq n} w_i}{s} \right) \right\rfloor$$

In order to maximize $\prod_{u \in \text{alloc}} \text{fp}_u$, we need to use as many processors as possible since $\text{fp}_u \leq 1$ for $1 \leq u \leq m$.

If one of the most reliable processors is not used, we can exchange it with a less reliable one, and thus increase the value of the product, so the formula is maximized when using the k most reliable processors, which is represented in Algorithm 1.

The second problem is expressed below:

$$\begin{aligned} & \text{MINIMIZE } |\text{alloc}| \frac{\delta_0}{b} + \frac{\sum_{1 \leq i \leq n} w_i}{s} + \frac{\delta_n}{b}, \\ & \text{UNDER THE CONSTRAINT} \end{aligned} \quad (4)$$

$$1 - (1 - \prod_{u \in \text{alloc}} \text{fp}_u) \leq \mathcal{FP}$$

Latency increases when $|\text{alloc}|$ is large, thus we need to find the smallest number of processors which satisfies constraint (4). As before, if one of the most reliable processors is not used, we can exchange it and improve the reliability without increasing the latency, which might lead to add fewer processors to the replication set for an identical reliability. Algorithm 2 thus returns the optimal solution. □

```

begin
  Find  $k$  minimum, such that
  
$$1 - (1 - \text{fp}^k) \leq \mathcal{FP}$$

  Replicate the whole pipeline as a single interval onto  $k$  processors.
end

```

Algorithm 2: *Fully Homogeneous* platforms: Minimizing \mathcal{L} for a fixed \mathcal{FP}

Remark Algorithm 2 is optimal as well in the case of heterogeneous failure probabilities. We add the most reliable processors to the replication scheme (thus increasing latency and decreasing the failure probability) while \mathcal{FP} is not reached.

3.3 Bi-criteria problems on *Com. Homogeneous* platforms

For *Communication Homogeneous* platforms, we first consider the simpler case where all failure probabilities are identical, denoted by *Failure Homogeneous*. In this case, the optimal bi-criteria solution still consists of the mapping of the pipeline as a single interval.

Theorem 4. *On Communication Homogeneous platforms with Failure Homogeneous, the solution to the bi-criteria problem can be found in polynomial time using Algorithm 3 or 4.*

Informally, we add the fastest processors to the replication set while the latency is not exceeded (or until \mathcal{FP} is reached), thus reducing the failure probability and increasing the latency.

```

begin
  Order processors in non-decreasing order of  $s_j$ ;
  Find  $k$  maximum, such that
    
$$k \times \frac{\delta_0}{b} + \sum_{1 \leq j \leq n} \frac{w_j}{s_k} + \frac{\delta_n}{b} \leq \mathcal{L}$$

  Replicate the whole pipeline as a single interval onto the fastest  $k$  processors;
  // Note that at any time  $s_k$  is the speed of the slowest processor used
  // in the replication scheme.
end

```

Algorithm 3: *Communication Homogeneous* platforms - *Failure Homogeneous* : Minimizing \mathcal{FP} for a fixed \mathcal{L}

```

begin
  Find  $k$  minimum, such that
    
$$1 - (1 - \text{fp}^k) \leq \mathcal{FP}$$

  Replicate the whole pipeline as a single interval onto the fastest  $k$  processors;
end

```

Algorithm 4: *Communication Homogeneous* platforms - *Failure Homogeneous* : Minimizing \mathcal{L} for a fixed \mathcal{FP}

Proof. (**Theorem 4**)

In this particular setting, Lemma 1 still applies, so we restrict to mappings as a single interval, and search for the optimal set of processors alloc which should be used.

The first problem is expressed as:

$$\begin{aligned}
 & \text{MINIMIZE } 1 - (1 - \text{fp}^{|\text{alloc}|}), \\
 & \text{UNDER THE CONSTRAINT} \\
 & |\text{alloc}| \frac{\delta_0}{b} + \frac{\sum_{1 \leq i \leq n} w_i}{\min_{u \in \text{alloc}} s_u} + \frac{\delta_n}{b} \leq \mathcal{L}
 \end{aligned} \tag{5}$$

The failure probability is smaller when $|\text{alloc}|$ is large, thus we need to add as many processors as we can while satisfying the constraint. The latency increases when adding more processors, and it depends of the speed of the slowest processors. Thus, if the $|\text{alloc}|$ fastest processors are not used, we can exchange a fastest processor with a used one without increasing latency. Algorithm 3 thus returns an optimal mapping.

The other problem is similar, with the following expression:

$$\begin{aligned} & \text{MINIMIZE } |\text{alloc}| \frac{\delta_0}{b} + \frac{\sum_{1 \leq i \leq n} w_i}{\min_{u \in \text{alloc}} s_u} + \frac{\delta_n}{b}, \\ & \text{UNDER THE CONSTRAINT} \end{aligned} \tag{6}$$

$$1 - (1 - \text{fp}^{|\text{alloc}|}) \leq \mathcal{FP}$$

We can thus find the smallest number of processors that should be used in order to satisfy \mathcal{FP} , and then use the fastest processors to optimize latency, which is done by Algorithm 4. \square

However, the problem is more complex when we consider different failure probabilities (*Failure Heterogeneous*). It is also more natural since we have different processors and there is no reason why they would have the same failure probability. Unfortunately for *Failure Heterogeneous*, we can exhibit for some instances of the problem an optimal solution in which the pipeline stages must be divided in several intervals. The complexity of the problem remains open, but we conjecture it is NP-hard.

3.4 Bi-criteria problems on *Fully Heterogeneous* platforms

For *Fully Heterogeneous* platforms, we restrict to heterogeneous failure probabilities, which is the most natural case. While both mono-criterion problems have a polynomial complexity, we prove that the bi-criteria problems are NP-hard.

Theorem 5. *On Fully Heterogeneous platforms, the bi-criteria (decision problems associated to the) optimization problems are NP-hard.*

Proof. (**Theorem 5**)

We consider the following decision problem on *Fully Heterogeneous* platforms: given a failure probability threshold \mathcal{FP} and a latency threshold \mathcal{L} , is there a mapping of failure probability less than \mathcal{FP} and of latency less than \mathcal{L} ? The problem is obviously in NP: given a mapping, it is easy to check in polynomial time that it is valid by computing its failure probability and latency.

To establish the completeness, we use a reduction from 2-PARTITION [10]. We consider an instance \mathcal{I}_1 of 2-PARTITION: given m positive integers a_1, a_2, \dots, a_m , does there exist a subset $I \subset \{1, \dots, m\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Let $S = \sum_{i=1}^m a_i$.

We build the following instance \mathcal{I}_2 of our problem: the pipeline is composed of a single stage with $w = 1$, and the input and output communication costs are $\delta_0 = \delta_1 = 1$. The platform consists in m processors with speeds $s_j = 1$ and failure probability $\text{fp}_j = e^{-a_j}$, for $1 \leq j \leq m$ (thus $0 \leq \text{fp}_j \leq 1$). Bandwidth are defined as $\text{b}_{\text{in},j} = 1/a_j$ and $\text{b}_{j,\text{out}} = 1$ for $1 \leq j \leq m$.

We ask whether it is possible to realize a latency of $S/2 + 2$ and a failure probability of $e^{-S/2}$. Clearly, the size of \mathcal{I}_2 is polynomial (and even linear) in the size of \mathcal{I}_1 . We now show that instance \mathcal{I}_1 has a solution if and only if instance \mathcal{I}_2 does.

Suppose first that \mathcal{I}_1 has a solution. The solution to \mathcal{I}_2 which replicates the stage on the set of processors I has a latency of $S/2 + 2$, since the first communication requires to sum $\delta_0/b_{in,j}$ for all processor P_j included in the replication scheme, and then both computation and the final output require a time 1. The failure probability of this solution is $1 - (1 - \prod_{j \in I} fp_j) = e^{-\sum_{j \in I} a_j} = e^{-S/2}$. Thus we have solved \mathcal{I}_2 .

On the other hand, if \mathcal{I}_2 has a solution, let I be the set of processors on which the stage is replicated. Because of the latency constraint,

$$\sum_{j \in I} \frac{1}{b_{in,j}} + 1 + 1 \leq \frac{S}{2} + 2$$

Since $b_{in,j} = 1/a_j$, this implies that $\sum_{j \in I} a_j \leq S/2$. Next we consider the failure probability constraint. We must have

$$1 - (1 - \prod_{j \in I} fp_j) \leq e^{-\frac{S}{2}}$$

and thus $e^{-\sum_{j \in I} a_j} \leq e^{-S/2}$, which forces $\sum_{j \in I} a_j \geq S/2$. Thus $\sum_{j \in I} a_j = S/2$ and we have a solution to the instance of 2-PARTITION \mathcal{I}_1 , which concludes the proof. \square

4 Related work and conclusion

In this paper, we have assessed the complexity of trading between response time and reliability, which are among the most important criteria for a typical user. Indeed, in the context of large scale distributed platforms such as clusters or grids, failure probability becomes a major concern [9, 11, 8], and the bi-criteria approach tackled in this paper enables to provide robust solutions while fulfilling user demands (minimizing latency under some reliability threshold, or the converse). We have shown that the more heterogeneity in the target platforms, the more difficult the problems. In particular, the bi-criteria optimization problem is polynomial for *Fully Homogeneous*, NP-hard for *Fully Heterogeneous* and remains an open problem for *Communication Homogeneous*.

Several other bi-criteria optimization problems have been considered in the literature. For instance optimizing both latency and throughput is quite natural, as these objectives represent trade-offs between user expectations and the whole system performance. See [15, 3, 4] for pipeline graphs and [17] for general application DAGs. In the context of embedded systems, energy consumption is another important objective to minimize. Three-criteria optimization (energy, latency and throughput) is discussed in [18].

For large scale distributed platforms such as production grids, throughput is a very important criterion as it measures the aggregate rate of processing of data, hence the global rate at which execution progresses. We can envision two types of replication: the first type is to replicate the same computation on different processors, as in this paper, to increase reliability. The second type is to allocate the processing of different data sets to different processors (say in a round-robin fashion), in order to increase the throughput. Both replication types can be conducted simultaneously, at the price of more resource consumption. Our future work will be devoted to the study of the interplay between throughput, latency and reliability, a very challenging algorithmic problem.

References

- [1] J. Abawajy. Fault-tolerant scheduling policy for grid computing systems. In *International Parallel and Distributed Processing Symposium IPDPS'2004*. IEEE Computer Society Press, 2004.
- [2] S. Albers and G. Schmidt. Scheduling with unexpected machine breakdowns. *Discrete Applied Mathematics*, 110(2-3):85–99, 2001.
- [3] A. Benoit, V. Rehn, and Y. Robert. Complexity results for throughput and latency optimization of replicated and data-parallel workflows. In *HeteroPar'2007: International Conference on Heterogeneous Computing, jointly published with Cluster'2007*. IEEE Computer Society Press, 2007.
- [4] A. Benoit, V. Rehn-Sonigo, and Y. Robert. Multi-criteria scheduling of pipeline workflows. In *HeteroPar'2007: International Conference on Heterogeneous Computing, jointly published with Cluster'2007*. IEEE Computer Society Press, 2007.
- [5] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *ICDCS'99 19th International Conference on Distributed Computing Systems*, pages 15–24. IEEE Computer Society Press, 1999.
- [6] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63:251–263, 2003.
- [7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
- [8] A. Duarte, D. Rexachs, and E. Luque. A distributed scheme for fault-tolerance in large clusters of workstations. In *NIC Series, Vol. 33*, pages 473–480. John von Neumann Institute for Computing, Julich, 2006.
- [9] A. H. Frey and G. Fox. Problems and approaches for a teraflop processor. In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 21–25. ACM Press, 1988.
- [10] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [11] A. Geist and C. Engelmann. Development of naturally fault tolerant algorithms for computing on 100,000 processors. <http://www.csm.ornl.gov/~geist/Lyon2002-geist.pdf>, 2002.
- [12] T. Saif and M. Parashar. Understanding the behavior and performance of non-blocking communications in MPI. In *Proceedings of Euro-Par 2004: Parallel Processing*, LNCS 3149, pages 173–182. Springer, 2004.
- [13] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Science Press, 1995.

- [14] J. Subhlok and G. Vondran. Optimal mapping of sequences of data parallel tasks. In *Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP'95*, pages 134–143. ACM Press, 1995.
- [15] J. Subhlok and G. Vondran. Optimal latency-throughput tradeoffs for data parallel pipelines. In *ACM Symposium on Parallel Algorithms and Architectures SPAA '96*, pages 62–71. ACM Press, 1996.
- [16] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2000.
- [17] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddyappan, and J. Saltz. An approach for optimizing latency under throughput constraints for application workflows on clusters. Research Report OSU-CISRC-1/07-TR03, Ohio State University, Columbus, OH, Jan. 2007. Available at <ftp://ftp.cse.ohio-state.edu/pub/tech-report/2007>.
- [18] R. Xu, R. Melhem, and D. Mosse. Energy-aware scheduling for streaming applications on chip multiprocessors. In *the 28th IEEE Real-Time System Symposium (RTSS'07), Tucson, Arizona*, December 2007.