N° d'ordre : — *N° attribué par la bibliothèque : ——*

- ÉCOLE NORMALE SUPÉRIEURE DE LYON -
Laboratoire de l'Informatique du Parallélisme

THÈSE

en vue d'obtenir le grade de

Docteur de l'Université de Lyon - École normale supérieure de Lyon

Spécialité : Informatique

au titre de l'École Doctorale Informatique et Mathématiques

présentée et soutenue publiquement le 6 septembre 2011 par

Fanny Dufossé

Scheduling for Reliability : Complexity and Algorithms

Directeur de thèse :	Yves	ROBERT	
Co-encadrante de thèse :	Anne	BENOIT	
Après avis de :	Arnold	ROSENBERG	Rapporteur
	Jean-Michel	FOURNEAU	Rapporteur

Devant la commission d'examen formée de :

Anne	BENOIT	Membre
Jean-Michel	FOURNEAU	Rapporteur
Alain	GIRAULT	Membre
Claire	HANEN	Membre
Yves	ROBERT	Membre
Arnold	ROSENBERG	Rapporteur

Acknowledgements

I would first thanks my two reviewers, Jean-Michel Fourneau and Arnold Rosenberg, for their work, my committee chairman, Claire Hanen, and Alain Girault for joining my thesis committee. They provide me insightful remarks and interesting research directions.

I particularly would like to thank my advisors, Anne Benoit and Yves Robert, for their numerous advices and lessons. Thank you for taking as much time and energy for teaching me the work of researcher, and particularly to design and write scientific papers.

I finally would like to thanks my family and my friends dor their support all through my PhD. Thank you for providing me encouragements, advices and some chocolate.

Contents

Chapter 1

Introduction

Since the advent of the first computers, processor speed has continually increased. However, users have always pursued their quest for more computing power than the fastest sequential machine can provide, and this computing power can only be provided through parallel solutions. Computations are split into smaller tasks that are executed on a platform composed of many processors. Research issues in this area mainly deal with the problems of splitting a given application into several tasks, and of allocating these tasks onto processors. These problems are referred to as *scheduling* problems.

A scheduling problem is generally composed of an application model, a platform model, a set of rules to allocate tasks to processors, and one or more criteria for comparing possible executions. Classical scheduling problems usually target execution time (or *latency*) minimization as unique objective [15]. A schedule indicates the execution graph, the allocation of tasks to processors, and for any processor, the order of execution of all tasks allocate on this processor and the dates of beginning and of completion of these tasks. The task graph of a schedule is the graph describing for any task, the set of tasks from which it receives data, and the set of tasks to which it sends its outputs. In most cases, execution dates can be deduced from the allocation and the execution order of the tasks. These two informations are referred to as a *mapping*. Tasks may have dependencies, and one speaks of a dependence graph that has to be scheduled. All the dependence constraints have to be enforced in the execution graph. Because the dependencies have to be respected, only some tasks can be executed at a given time-step, while others have to wait for some output. This considerably increases the difficulty of the scheduling problem. It is well known that simple instances of this classical optimization problem are already NP-hard, even in the context of same speed processors.

A classical scheduling problem concerns data stream applications. Streaming applications are widely used to model multimedia applications or real time data processing [83, 41]. Such applications are classically modeled by workflows. A workflow is given by a task graph: a weighted directed acyclic graph (DAG), with nodes representing tasks, and edges modeling first-in-first-out channels. Each data set is input into the graph using input channel(s) and the outputs are produced on the output channel(s). Consecutive data sets continually flow through these applications. The problem of scheduling a workflow graph on a platform is widely studied [21, 75, 84]. Many criteria can be considered: in addition to the *latency* minimization, a typical objective is the *throughput* maximization, (or equivalently the *period* minimization, where the *period* is defined as the inverse of the *throughput*). The period is the interval of time between the completions of two consecutive data sets. Other criteria can be considered, such as energy consumption, or reliability of the computations. A well-studied subclass of these applications is that of linear chains of tasks, or *linear workflows* [73, 74].

The task graph of a workflow has a huge impact on its possible schedules. It is therefore interesting to consider a similar model with the possibility to choose the structure of the execution graph. The

model of filtering applications [16, 71] is a variant to workflows where the schedule has to decide the order of execution of tasks and the execution graph. In this model, each task modifies the size of its input data by a fixed ratio, increasing or decreasing the size of data. This ratio is named the *selectivity* of the task. Moreover, the execution time of a task is proportional to its input size. Therefore, the set of predecessors of a task affects its execution time. As for workflows, a precedence graph is given, but other edges can be added to this graph to obtain an execution graph, with the only constraint that this graph remains acyclic. A filtering application with a given execution graph is equivalent to a workflow.

As for the application, the target platform needs to be carefully described. A platform model provides the speed of processors, the communication model and potentially some other properties of processors (energy consumption, reliability, data storage capacity,...). It can also describe a hierarchy between processors. All processors can be considered independently, but the model can assume a master-worker paradigm: the master 'decides' for the schedule, and communicates with workers, and workers receive tasks, execute them, and transmit the outputs of tasks to the master. We generally consider that the processors of the platform are fully connected, with links of homogeneous bandwidth capacity. It means that any pair of processors is connected, or that all processors are connected to a same switch. We consider both homogeneous and heterogeneous platforms, that means platforms of same speed processors or with processors of different speeds.

An important issue of platform models concerns communications. Modern processors are *multi-port* and can simultaneously communicate with many processors. However, the whole communication process is not parallel. For example, even in a multi-port processor, the communications with the network use a single device, namely the network card. Then the *one-port* model [12] and the bounded *multi-port* [47] model both make sense. The *multi-port* considers that bandwidths of all ports of the processor are independent. We generally consider that a *multi-port* processor can communicate with another processor during a computation, where a *one-port* processor is not able to overlap these two operations. We assume that the time needed to send a message from one processor to another is directly proportional to the size of the message. More precise models propose an affine function for the communication time, but for large enough messages, a linear function is adequate. The communication time is then the ratio of the size of the message and the bandwidth of the processor. For master-worker platforms, we can suppose that the master has a larger bandwidth than the workers.

In the previous models, we consider that the platform is dedicated to a single user. But many organizations propose to end-users to offer CPU time on computers of volunteers during idle times, to the benefit of research projects. Such a model, named a *desktop grid* [13, 19], is characterized by its instability. A computing resource can be removed or slowed down at any time by its owner.

Such platform model highlights the problem of reliability of processors. Even on classical platforms, processors are subject to different types of failures. The two main failure types are *transient* and *fail-stop* failures. *Transient* failures correspond to arithmetic/software errors or recoverable hardware faults (power loss). These failures are considered as instantaneous: a *transient* failure only impacts the current task (if any) on the concerned processor. *Transient* failures are the most common failures in modern processors [62]. Communications, as computations, are subject to such failures. Unlike *transient* failures, when a *fail-stop* failure occurs, the concerned processor fails down, and remains unavailable for an undetermined interval of time.

In order to provide efficient schedules for reliability, failure occurrences need to be modeled by realistic probability laws. Concerning *transient* failures, a realistic model uses Poisson law, which means that we consider that the failure rate per time unit is constant. Modern fail-silent hardware components have a failure rate around 10^{-6} per hour. This law does not take into account the aging of processors. Unlike *transient* failures, *fail-stop* failures are modeled with discrete time. A classical probability model for *fail-stop* failures is the Markov process. It supposes that the state at next time slot only depends on

the current state. In practice, the Markov assumption is not valid in real life traces (the history of states in real life platforms). More accurate distribution models are used to simulate processor behavior, as Weibull distributions or Pareto distributions [88].

For a given application model and a given platform model, many mapping rules can make sense. Three types of mappings are generally studied: *general mappings*, *interval mappings* and *one-to-one mappings*. In a *general mapping*, each task can be allocated on each processor, independently of its predecessors and successors. Then, a schedule needs to order the tasks computed on a same processor and the communications between processors. A *one-to-one mapping* allocates at most one task by processor. The schedule trivially results from the mapping. The *interval mapping* is only defined for linear applications. In this model, the chain of tasks is split in intervals that are allocated one by one to processors. As in *one-to-one mappings*, the schedule trivially results from the mapping.

The possible schedules are compared with one or more criteria: period, latency, reliability, and so on. We consider here many multi-criteria problems. Different methods can be used to study such problems:

1. The lexicographic ordering [33]: The criteria are ordered, and the best solution is the best for first criterion, the best for second criterion with respect to the previous rule, and so on.
2. The aggregate objective approach [87]: A real function is requested with input all criteria of the problem. We obtain a single criterion that is for each solution the value of this function. In most cases, this function is linear.
3. The ϵ-constraints [57]: a main objective is selected, and a bound is decided for any other criterion. The optimal solution is the best one for the first criterion, meeting the bounds for the other ones.
4. The Pareto optimality [65]: This method gives all Pareto optimal solutions. A solution S_1 Pareto dominates a solution S_2 if for any criterion c, solution S_1 is better than solution S_2 for criterion c. Then, a solution S is Pareto optimal if no other solution Pareto dominates solution S.

The first three methods give Pareto optimal solutions. The first one however only provides few ones, while any Pareto optimal solution can be obtained with methods 2 and 3, with the appropriate aggregate function or the appropriate bounds. In the lexicographic ordering method, only one criterion is considered as really interesting. The aggregative method consists in transforming a multi-criteria problem into a mono-criterion one. All criteria are considered in the aggregate function, but the importance and the difference of the different criteria cannot appear clearly in a single function, all the less in a linear function. Therefore, for our problems, the ϵ-constraints method is the more natural approach, and we use it in most cases. A bound on all criteria except one describes the minimum accepted result, and the optimal solution is the solution meeting the previous bounds, that minimizes the last criterion.

In the following, we detail the structure and contribution of this work. This thesis is composed of four main chapters.

Mapping filtering streaming applications

The schedule of a filtering application consists of an execution graph and a mapping of tasks onto processors. Our study addresses the problem of one-to-one mapping filtering applications on homogeneous and heterogeneous platforms. The optimization criteria are the period and the latency.

In a first part, we consider the problem on homogeneous and heterogeneous platforms, without communication costs, to optimize period, latency or both criteria. For each variant on homogeneous platforms, we present an optimal polynomial time algorithm, and for each variant on heterogeneous platform, we prove its NP-completeness, and its inapproximability unless $P = NP$.

In a second part, three communication models are described: a bounded multi-port model with overlap, and two one-port models without overlap, respecting the order of data set, or not. These models are

applied to homogeneous platforms with one-to-one mappings. With these models, for a fixed mapping, computing the best execution dates becomes difficult. Scheduling a filtering application with fixed execution graph is equivalent to scheduling a workflow. We prove the NP-completeness of some variants of this problem. We also present NP-completeness proofs for all variants of the problem of computing the best execution graph.

This study is finally extended to general mappings on linear heterogeneous platforms. The execution order of tasks can be fixed or not, with arbitrary costs or with costs proportional to the inverse of processors speed, and with or without communication costs. The variant with a fixed order of tasks correspond to an interval mapping problem. The complexity of all variants are established.

Reliability and performance optimization of pipelined real-time systems

In a classical failure model, a given computation is successful with a certain probability. This model is similar to the filtering model, with success probability replacing selectivity. In order to increase the reliability of computations, we use duplication: many processors execute the same computations, so that if one of these processors fails, the computation can continue on the other ones. We consider, in this study, linear workflows with interval mappings: the chain of tasks is split into intervals, and each interval is allocated to one or many processors. A processor computes at most one interval. Communications are transmitted using routing operations: all output communications of a given interval are transmitted to a same processor which sends it to processors that compute the next interval as soon as the first communication is completed. Because of the bandwidth capacity of this routing processor, the number of communications in parallel, and therefore the number of replications of a given interval is bounded.

Linear time algorithms are described to compute the expected period and the expected latency of a given mapping. Three criteria are considered in this study: period, latency and reliability. The complexity of mono-criterion and multi-criteria problems is given for homogeneous and heterogeneous platforms. A greedy algorithm is described to optimally allocate a given set of intervals on a homogeneous platform. Two heuristics are described and compared on homogeneous and heterogeneous platforms.

Scheduling parallel iterative applications on volatile resources

Transient failures are the most common failures in classical modern grids. We study the mapping of iterative applications on this platform model: at each iteration, a set of identical tasks is executed. Then, a synchronization of processors occurs, and a new iteration begins. A set of data is needed for any task, and all tasks execute the same program. In this chapter, we consider independent tasks. Processors can be *up*, *down* or *reclaimed* by their owners. When failing, a processor loses all data and program. If it becomes reclaimed, its computation and communication are interrupted until it is up again. We use a three-state Markov chain to model the state of processors. This chapter first studies the complexity of the off-line problem. We prove the NP-completeness of this problem and provide an inapproximability result. Closed-form formulas are established for the probability of success of a task on a processor, and its expected completion time. Heuristics are provided to solve the on-line problem, and their performances are assessed through extensive simulations.

Scheduling parallel coupled iterative applications on volatile resources

The same model is extended to tightly coupled tasks: all tasks have to progress at same speed. If some tasks are not allocated, no computation can be done. If some tasks are allocated on a reclaimed processor, all tasks are interrupted until all concerned processors become available again. If some processor computing at least one task fails, then all executed work for the current iteration is lost. These

constraints make the computation more unreliable than in the previous study. The NP-completeness of the off-line problem is proved, and optimal algorithms are provided for some polynomial particular instances. In this new setting of tightly coupled tasks, we cannot consider the probability of success of tasks independently. A polynomial-time approximation scheme is provided to compute the probability of success of a schedule, and its expected completion time. Heuristics are provided and compared through simulations.

Chapter 2

Mapping filtering streaming applications

2.1 Introduction

Pipelined workflows are often used to model streaming applications, such as video and audio encoding and decoding, digital signal processing (DSP) applications, etc. [21, 75, 84]. A workflow graph contains several *nodes*, and these nodes are connected to each other using first-in-first-out *channels*. Data is input into the graph using input channel(s) and the outputs are produced on the output channel(s). Since this model represents a large class of important applications, the problem of mapping and scheduling these applications on distributed platforms is well studied in the literature. The goal is to map each node onto some processor so as to optimize some scheduling objectives. Since data continually flow through these applications, typical objectives of the scheduler are *throughput* maximization (or equivalently *period* minimization, where the period is defined as the inverse of the throughput) and/or *latency* (also called response time) minimization [77, 80, 78, 79, 10, 83].

This chapter addresses a related problem of mapping and scheduling applications where each node may *filter* data, i.e., increase or decrease the size of its input data set by a fixed ratio. This problem models query optimization over web services [71, 16]. In the query optimization problem, as in the pipelined workflow problem, we have a collection of various services that must be applied on a stream of consecutive data sets. As with workflows, we have a graph with nodes (the services) and precedence edges (dependence constraints between services), with data flowing continuously from the input node(s) to the output node(s). We aim at one-to-one mappings, that means that any processor will compute at most one task. The goal is to map each service onto a processor, or server, so as to optimize the same objectives as before (period and/or latency).

The problem is made more difficult due to the following problems. First, services can *filter* the data by a certain amount, according to their *selectivity*. Consider a service C_i with selectivity σ_i: if the incoming data is of size δ, then the outgoing data will be of size $\delta \times \sigma_i$. The initial data is of size δ_0. We see that the data is shrunk by C_i (hence the term "filter") when $\sigma_i < 1$ but it can also be expanded if $\sigma_i > 1$. Second, the filtering affects the computation costs of each service. Each service C_i has an absolute cost w_i, which represents time needed to process a data set of size δ_0 with a server of speed 1. Therefore, the cost of the service on a server S_u of speed s_u is $C_{i,u} = w_i/s_u$. But the real cost of computation is proportional to the actual size of the input data, which may have been shrunk or expanded by the predecessors of C_i in the mapping. Therefore, if the predecessors of C_i shrink or expand the data set by the factor of σ, the size of input data set into C_i is $\sigma \times \delta_0$ and the time to execute this data set when service C_i is mapped onto server S_u is $\sigma \times C_{i,u}$. Third, and most important, as opposed to the pipelined workflow problem, the solution is allowed to add additional edges to the precedence graph. Note that we consider that the set of processors is fully interconnected. Intuitively, for services

of selectivities less than 1, we can consider that any service removes some lines of the original file, and the result file is the intersection of the lines kept by all services. Then, the order of execution of these services has no impact on the result. Adding additional edges to the precedence graph does not change the result of the application and can modify the execution time of schedules.

For example, consider two arbitrary services C_i and C_j. If there is a precedence constraint from C_i to C_j, we need to enforce it. But if there is none, meaning that C_i and C_j are independent, we may still introduce a (fictitious) edge, say from C_j to C_i, in the mapping, meaning that the output of C_j is fed as input to C_i. If the selectivity of C_j is small ($\sigma_j < 1$), then it shrinks each data set, and C_i will operate on data sets of reduced volume. As a result, the cost of C_i will decrease in proportion to the volume reduction, potentially leading to a better solution. Basically, there are two ways to decrease the final cost of a service: (i) map it on a fast server; and (ii) map it as a successor of a set of services with small selectivities.

As already pointed out, period and latency are both very important objectives. The inverse of the period (the throughput) measures the aggregate rate of processing of data, and it is the rate at which data sets can enter the system. 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. Minimizing the latency is antagonistic to minimizing the period, and tradeoffs should be found between these criteria. Efficient mappings aim at the minimization of a single criterion, either the period or the latency, but they can also use a bi-criteria approach, such as minimizing the latency under period constraints (or the converse). The main objective of this work is to assess the complexity of the previous optimization problems, with identical servers or with different-speed servers, with and without communication costs, in the case of one-to-one or general mappings. All our hypotheses are those of Srivastava et al. [70, 71, 16].

In general, we have to organize the execution of the application by assigning a server to each service and by deciding which service will be a predecessor of which other service (therefore building an execution graph, or *plan*), with the goal of minimizing the objective function. The edges of the execution graph must include all the original dependence edges of the application. We are free to add more edges if it decreases the objective function. Note that the selectivity of a service influences the execution time of *all* its successors, if any, in the mapping. For example if three services C_1, C_2 and C_3 are arranged along a linear chain, as in Figure 2.1, then the cost of C_2 is $\sigma_1 w_2$ and the cost of C_3 is $\sigma_1\sigma_2 w_3$. If C_i is mapped onto S_i, for $i = 1, 2, 3$, then the period is $\mathcal{P} = \max\left(\frac{w_1}{s_1}, \frac{\sigma_1 w_2}{s_2}, \frac{\sigma_1\sigma_2 w_3}{s_3}\right)$, while the latency is $\mathcal{L} = \frac{w_1}{s_1} + \frac{\sigma_1 w_2}{s_2} + \frac{\sigma_1\sigma_2 w_3}{s_3}$. Here, we also note that selectivities are independent: for instance if C_1 and C_2 are both predecessors of C_3, as in Figure 2.1 or in Figure 2.2, then the cost of C_3 becomes $\sigma_1\sigma_2 w_3$. In term of probabilities, it means that, for a given line of a data file, the probability to be kept by C_1 is independent of the probability to be kept by C_2. With the mapping of Figure 2.2, the period is $\mathcal{P} = \max\left(\frac{w_1}{s_1}, \frac{w_2}{s_2}, \frac{\sigma_1\sigma_2 w_3}{s_3}\right)$, while the latency is $\mathcal{L} = \max\left(\frac{w_1}{s_1}, \frac{w_2}{s_2}\right) + \frac{\sigma_1\sigma_2 w_3}{s_3}$. We see from the latter formulas that the model neglects the cost of *joins* when combining two services as predecessors of a third one. Indeed, we make the hypothesis that the cost of join operations is negligible in front of the service costs, similarly to [71, 16].

While the computation of the period and latency of a given mapping is easy in a model without communication cost, it turns out surprisingly difficult when adding such costs into the story. The mapping does not determine the execution completely in this case. For example, in Figure 2.3, we do not know if service C_1 will send the result of its computation first to C_2 or to C_4, and in which order C_5 will receive its data. The orchestration of the communications has a dramatic impact on the value of the period and latency. Suppose that the value of all computation costs is 4, and that each communication cost is 1. In this case, the path $C_1 \to C_2 \to C_3 \to C_5$ is longer than the path $C_1 \to C_4 \to C_5$. In order to reach the

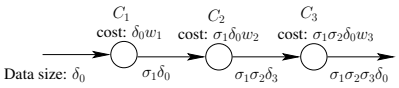

Figure 2.1: Chaining services.

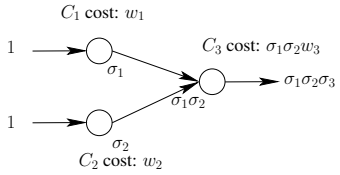

Figure 2.2: Combining selectivities.

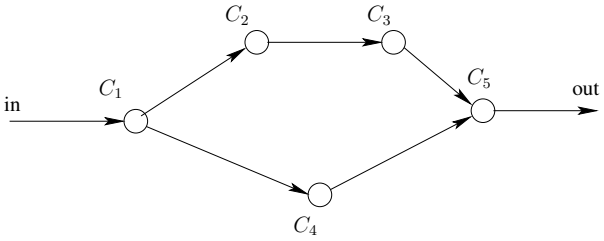

Figure 2.3: Example with communications.

optimal period, we will thus have to "share" the idle time between C_1, C_4 and C_5, or to execute other operations during this idle time. For each communication $C_i \to C_j$, we have to find a time-step that is well suited for both source and sink processors. We come back to this example in Section 2.4.4.

In this chapter, we identify three realistic communication cost models. The first two models sequentialize the operations of a given processor, while the third model allows for communication/computation overlap, and for (a limited number of) simultaneous sends or receives per processor. In all cases, the plan cannot reduce to the execution graph any longer. A complete description of the mapping now requires an *operation list*, which provides the time-steps at which each computation and communication begins and ends. Communication costs induce an additional level of difficulty: given an execution graph, it is impossible without the operation list to determine how to orchestrate the operations so as to achieve the best period and latency. Indeed, we prove in Section 2.4 that in most cases, the problem of computing the optimal operation list for a given plan is NP-complete.

In this chapter, we establish several new and important complexity results about the *evaluation problems* (given the mapping, determine best period or latency) and the *optimization problems* (determine optimal mapping):

- For the model without communication cost, the evaluation problems are easy. We introduce an optimal polynomial algorithm for the latency minimization problem on a homogeneous platform. This result nicely complements the corresponding result for period minimization, that was shown to have polynomial complexity in [71]. We also show the polynomial complexity of the bi-criteria problem (minimizing latency while not exceeding a threshold period). Moving to heterogeneous resources, we prove the NP-completeness of both the period and latency minimization problems, even for independent services. Therefore, the bi-criteria problem also is NP-complete in this case. Furthermore, we prove that there exists no constant factor approximation algorithms for these NP-hard problems unless P=NP.
- For the models with communication costs, our main result is that computing the period or the latency in all these models turns out to be difficult. First, the optimization problems are all NP-hard on homogeneous machines, while they are polynomial when we do not model communication [71, 16]. Therefore, modeling communication costs explicitly has a huge impact on the

difficulty of mapping filtering services. In addition, and quite unexpectedly, the evaluation problems (given a plan, find the optimal operation list) also are of combinatorial nature. Finally, the choice of the communication model has a tremendous impact on the values that can be achieved. Many of our results and counter-examples apply to regular workflows (without selectivities), and should be of great interest to the whole community interested in scheduling streaming applications.

– We extend the results of [70] in another important direction: we investigate the situation where servers are no longer independent but instead where they are ordered along a linear chain of precedence. We exhibit polynomial algorithms for problems with totally ordered tasks and proof of NP-completeness for problem with free order of tasks and arbitrary costs of tasks on processors. In the case of proportionnal costs of tasks, optimization of period is proved NP-complete and optimization of latency is proved polynomial.

The rest of this chapter is organized as follows. Section 2.2 is devoted to a survey of related work. We study the optimization problems without communication cost in Section 2.3. Next we present the different models of communication costs and corresponding complexity results in Section 2.4. Then Section 2.5 is devoted to problems with general mappings on a linear platforms. Finally we give some conclusions and perspectives in Section 2.6.

2.2 Related work

The main application of filtering services is query optimization over web services [70, 71, 16], an increasingly important application with the advent of Web Service Management Systems [31, 60]. Note that the approach also applies to general data streams [5] and to database predicate processing [18, 46].

In [71, 16], the authors consider the case where the web services should be mapped one-to-one onto identical servers without communication cost. Section 2.3 extends this model to heterogeneous platforms, and introduces the latency as criterion. Section 2.4 presents realistic communication models for this platform model and studies the mapping and scheduling problems of filtering application in this context.

In [70], the authors target the problem of mapping filtering application on a linear chain of processors of increasing speed. In Section 2.5, speed of processors is no more increasing. We also extend the results of [70] in another important direction: we investigate the situation where services are no longer independent but instead where they are ordered along a linear chain of precedence. In this case, both services and processors are arranged according to a fixed prescribed order. This problem is the extension of the well known chains-to-chains problem [61] to the case where nodes have a selectivity, and it has a great practical significance because linear dependence chains are ubiquitous in workflow applications (see [73, 74]).

In [1], the authors consider a set of jobs characterized by a certain success probability and a reward. The resulting problem is similar to our problem, but they maximize the reward while we minimize the cost. They present a polynomial algorithm in the case of a single server, and they prove that the problem becomes NP-complete when considering 2 servers.

2.3 Problems without communication cost

In this section, we study the complexity of the problems on homogeneous or heterogeneous platforms without any communication cost. First, we formally state the optimization problems, and then we present complexity results for homogeneous platforms and for heterogeneous platforms.

2.3.1 Framework

As stated above, the target application $\mathcal{A}$ is a set of services (or filters, or queries) linked by precedence constraints. We write $\mathcal{A} = (\mathcal{F}, \mathcal{G})$ where $\mathcal{F} = \{C_1, C_2, \ldots, C_n\}$ is the set of services and $\mathcal{G} \subset \mathcal{F} \times \mathcal{F}$ is the set of precedence constraints. If $\mathcal{G} = \emptyset$, we have independent services. A service C_i is characterized by its cost w_i and its selectivity σ_i.

For the computing resources, we have a set $\mathcal{S} = \{S_1, S_2, \ldots, S_p\}$ of servers. In the case of homogeneous platforms, servers are identical while in the case of heterogeneous platforms, each server S_u is characterized by its speed s_u. Let δ_0 be the size of input data. We always assume that there are more servers available than services (hence $n \leq p$), and we search a one-to-one mapping, or allocation, of services to servers. The one-to-one allocation function $alloc$ associates to each service C_i a server $S_{alloc(i)}$.

We also have to build a graph $G = (\mathcal{C}, \mathcal{E})$ that summarizes all precedence relations in the mapping. The nodes of the graph are couples $(C_i, S_{alloc(i)}) \in \mathcal{C}$, and thus define the allocation function. There is an arc $(C_i, C_j) \in \mathcal{E}$ if C_i precedes C_j in the execution. There are two types of such arcs: those induced by the set of precedence constraints $\mathcal{G}$, which must be enforced in any case, and those added to reduce the objective function. $\mathsf{Ancest}_j(G)$ denotes the set of all ancestors [1] of C_j in G, but only arcs from direct predecessors are kept in $\mathcal{E}$. In other words, if $(C_i, C_j) \in \mathcal{G}$, then we must have $C_i \in \mathsf{Ancest}_j(G)$ [2]. The graph G is called a plan. Given a plan G, the execution time of a service C_i is $C_{\text{exec}}(i) = \delta_0 \times \left(\prod_{C_j \in \mathsf{Ancest}_i(G)} \sigma_j\right) \times \frac{w_i}{s_{alloc(i)}}$. We note $L(i)$ the completion time of service C_i with the plan G, which is the length of the path from an entry node to C_i, where each node is weighted with its execution time. The period $\mathcal{P}$ is defined as the interval between the completion of consecutive data sets. With this definition, the system can process data sets at rate $1/\mathcal{P}$ (the throughput). In steady state, a new data set enters the system every $\mathcal{P}$ time units, and several data sets can be processed concurrently within the system. The latency (or response time) is the time needed to execute a single data set entirely. We can formally define the period $\mathcal{P}$ and latency $\mathcal{L}$ of a plan G in this model without communication cost:

$$\mathcal{P}(G) = \max_{(C_i, S_{alloc(i)}) \in \mathcal{C}} C_{\text{exec}}(i) \quad \text{and} \quad \mathcal{L}(G) = \max_{(C_i, S_{alloc(i)}) \in \mathcal{C}} L(i).$$

We can scale all costs as $w_k \leftarrow \delta_0 \times w_k$, allowing us to set $\delta_0 = 1$ without loss of generality.

In the following, we study three optimization problems: (i) MINPERIOD: find a plan G that minimizes the period; (ii) MINLATENCY: find a plan G that minimizes the latency; and (iii) BICRITERIA: given a bound on the period K, find a plan G whose period does not exceed K and whose latency is minimal. Each of these problems can be tackled, (a) either with an arbitrary precedence graph $\mathcal{G}$ (case PREC) or with independent services (case INDEP); and (b) either with identical servers ($s_u = s$ for all servers S_u, homogeneous case HOM), or with different-speed servers (heterogeneous case HET). For instance, MINPERIOD-INDEP-HOM is the problem of minimizing the period for independent services on a homogeneous platform while MINLATENCY-PREC-HET is the problem of minimizing the latency for arbitrary precedence constraints on a heterogeneous platform.

2.3.2 Homogeneous platforms

We investigate first the optimization problems with homogeneous resources. The problem MINPERIOD-PREC-HOM (minimizing the period with precedence constraints and identical resources) was

1. The ancestors of a service are the services preceding it, and the predecessors of their predecessors, and so on.
2. Equivalently, $\mathcal{G}$ must be included in the transitive closure of $\mathcal{E}$.

shown to have polynomial complexity in [71, 16]. We show that the problem MINLATENCY-PREC-HOM is polynomial too. Because the algorithm is quite complicated, we start with an optimal algorithm for the simpler problem MINLATENCY-INDEP-HOM. Although the polynomial complexity of the latter problem is a consequence of the former, it is insightful to follow the derivation for independent services before dealing with the general case. Finally, we propose optimal algorithms for BICRITERIA-INDEP-HOM and BICRITERIA-PREC-HOM.

Latency

We describe here optimal algorithms for MINLATENCY-HOM, starting with the case INDEP and then generalizing to the PREC case.

Algorithm 1 tackles the case INDEP. The idea is to schedule services step by step by increasing order of costs. At each step, the remaining service of lower cost is scheduled so as to minimize its completion time, considering the selectivities of services already scheduled. Theorem 2.1 states that this algorithm is optimal for problem MINLATENCY-INDEP-HOM.

Algorithm 1: Optimal algorithm for MINLATENCY-INDEP-HOM.

Data: n independent services with selectivities $\sigma_1, ..., \sigma_p \leq 1$, $\sigma_{p+1}, ..., \sigma_n > 1$, and ordered costs $w_1 \leq \cdots \leq w_p$
Result: a plan G optimizing the latency

```
G is the graph reduced to node C_1;
for i = 2 to n do
    for j = 0 to i − 1 do
        Compute the completion time L_j(C_i) of C_i in G with predecessors C_1, ..., C_j;
    end
    Choose j such that L_j(C_i) = min_k{L_k(C_i)};
    Add the node C_i and the edges C_1 → C_i, ..., C_j → C_i to G;
end
```

Theorem 2.1. (Independent services) *Algorithm 1 computes the optimal plan for* MINLATENCY-INDEP-HOM *in time* $O(n^2)$.

Proof. We show that Algorithm 1 verifies the following properties:
- (A) $L(1) \leq L(2) \leq \cdots \leq L(p)$;
- (B) $\forall i \leq n$, $L(i)$ is optimal.

Because the latency of any plan G' is the completion time of its last node (a node C_i such that $\forall C_j, L'(i) \geq L'(j)$), property (B) shows that $\mathcal{L}(G)$ is the optimal latency. We prove properties (A) and (B) by induction on i: for $1 \leq i \leq n$, we prove that $L(i)$ is optimal, and that for $1 \leq i \leq p$, $L(1) \leq L(2) \leq \cdots \leq L(i)$.

For $i = 1$, C_1 has no predecessor in G, so $L(1) = w_1$. Suppose that there exists G' such that $L'(1) < L(1)$. If C_1 has no predecessor in G', then $L'(1) = w_1 = L(1)$. Otherwise, let C_i be a predecessor of C_1 in G' such that C_i has no predecessor itself. $L'(1) > w_i \geq w_1$. In both cases, we obtain a contradiction with the hypothesis $L'(1) < L(1)$. So $L(1)$ is optimal.

Suppose that for a fixed $i \leq p$, $L(1) \leq L(2) \leq \cdots \leq L(i-1)$ and $\forall j < i$, $L(j)$ is optimal. Suppose that there exists G' such that $L'(i)$ is optimal. Let C_k be the predecessor of C_i of greatest cost in G'. If $w_k > w_i$, we can choose in G' the same predecessors for C_i than for C_k, thus strictly reducing $L'(i)$.

However, $L'(i)$ is optimal. So, we obtain a contradiction and $w_k \leq w_i$. Thus,

$$\begin{array}{rcll} L'(i) & = & L'(k) + \left(\prod_{C_j \in \mathsf{Ancest}L'(i)} \sigma_j\right) w_i & \\ & \geq & L'(k) + \left(\prod_{j \leq k} \sigma_j\right) w_i & \text{by definition of } C_k \\ & \geq & L(i) & \text{by construction of } G. \end{array}$$

Therefore, since $L'(i)$ is optimal by hypothesis, we have $L'(i) = L(i)$.

Suppose now that $L(i) < L(i-1)$. Then, C_{i-1} is not a predecessor of C_i in G. We construct G'' such that all edges are the same as in G except those oriented to C_{i-1}: predecessors of C_{i-1} will be the same as predecessors of C_i. We obtain

$$\begin{array}{rcll} L''(i-1) & = & \max_{k \leq j} L(k) + \prod_{k \leq j} \sigma_k w_{i-1} & \text{by construction of node } C_{i-1} \\ & \leq & \max_{k \leq j} L(k) + \prod_{k \leq j} \sigma_k w_i = L(i) & \end{array}$$

However, $L(i-1)$ is optimal, and so $L(i-1) \leq L''(i-1) \leq L(i)$, which leads to a contradiction. Therefore, $L(1) \leq L(2) \leq \cdots \leq L(i)$.

At this point, we have proved that the placement of all services of selectivity smaller than 1 (services $C_1, \ldots, C_p$) is optimal, and that $L(1) \leq L(2) \leq \cdots \leq L(p)$. We now proceed with services C_{p+1} to C_n.

Suppose that for a fixed $i > p$, $\forall j < i$, $L(j)$ is optimal. For all $k > p$, we have

$$\begin{array}{rcl} \max_{j \leq k} L(j) + \prod_{j \leq k} \sigma_j \times w_i & = & \max_{j=p}^{k} L(j) + \prod_{j=1}^{k} \sigma_j \times w_i \\ & \geq & L(p) + \prod_{j \leq k} \sigma_j \times w_i \\ & > & L(p) + \prod_{j \leq p} \sigma_j \times w_i \end{array}$$

This relation proves that in G, service C_i has no predecessor of selectivity strictly greater than 1. Suppose that there exists G' such that $L'(i)$ is optimal. Let C_k be the predecessor of C_i in G' of greatest cost. Then $\mathsf{Ancest}_i(G') \in \{1, k\}$ and, similarly for the case $i \leq p$, we obtain $L'(i) \geq L(i)$, and thus $L(i)$ is optimal.

■

Algorithm 2 generalizes the principles of Algorithm 1 in the model PREC. In this algorithm, at each step, the task scheduled is the one that can be scheduled with minimum possible completion time. Theorem 2.2 states its optimality.

Algorithm 2: Optimal algorithm for MINLATENCY-PREC-HOM.

Data: n services, a set $\mathcal{G}$ of dependence constraints
Result: a plan G optimizing the latency
1 G is the graph reduced to the node C of minimal cost with no predecessor in $\mathcal{G}$;
2 **for** $i = 2$ **to** n **do**
3 // At each step we add one service to G, hence the $n-1$ steps;
4 Let S be the set of services not yet in G and such that their set of predecessors in $\mathcal{G}$ is included in G;
5 **for** $C_j \in S$ **do**
6 **for** $C_k \in G$ **do**
7 Compute the set S' minimizing the product of selectivities among services of latency less than $L(k)$, and including all predecessors of C_j in $\mathcal{G}$ (using an algorithm from [16], whose execution time is $O(n^3)$);
8 **end**
9 Let S_j be the set that minimizes the latency of C_j in G and L_j be this latency;
10 **end**
11 Choose a service C_j such that $L_j = \min\{L_k, C_k \in S\}$;
12 Add to G the node C_j, and $\forall C_k \in S_j$, the edge $C_k \rightarrow C_j$;
13 **end**

Theorem 2.2. (General case) *Algorithm 2 computes the optimal plan for* MINLATENCY-PREC-HOM *in time* $O(n^6)$.

Proof. Let $\mathcal{A} = (\mathcal{F}, \mathcal{G})$ with $\mathcal{F} = \{C_1, C_2, \ldots, C_n)$ be an instance of MINLATENCY-PREC-HOM. Let G be the plan produced by Algorithm 2 on this instance, and services are renumbered so that C_i is the service added at step i of the algorithm. Then we prove by induction on i that $L(1) \leq L(2) \leq ... \leq L(n)$, and G is optimal for $L(i)$, $1 \leq i \leq n$. In the following, we say that a plan is *valid* if all precedence edges are included. The plan G is valid by construction of the algorithm.

By construction, C_1 has no predecessors in G. Therefore, $L(1) = w_1$. Let G' be a valid plan such that $L'(1)$ is optimal. If C_1 has no predecessors in G', then $L'(1) = L(1)$. Otherwise, let C_i be a predecessor of C_1 which has no predecessors in G'. G' is valid, thus C_i has no predecessors in $\mathcal{G}$. And by construction of G, we have $w_1 \leq w_i$. Therefore, $L'(1) \geq w_i \geq w_1 = L(1)$. Since $L'(1)$ is optimal, $L(1) = L'(1)$ and thus $L(1)$ is optimal.

Suppose that for a fixed $i \leq n$, we have $L(1) \leq L(2) \leq ... \leq L(i-1)$, and $\forall j < i, L(j)$ is optimal. Let us prove first that $L(i-1) \leq L(i)$. If C_{i-1} is a predecessor of C_i, then the result is true. Otherwise, and if $L(i-1) > L(i)$, then C_i would have been chosen at step $i-1$ of the algorithm (line 11) instead of C_{i-1}, which leads to a contradiction. It remains to prove that $L(i)$ is optimal. Let us consider a valid plan G' such that $L'(i)$ is optimal.
(i) Suppose first that C_i has at least one predecessor C_l with $l > i$ in G'. For such predecessors, at least one of them has its own set of predecessors included in $\{C_1, ..., C_{i-1}\}$. Let C_k be the service of maximal latency $L'(k)$ of the previous set of predecessors. Thus, $k > i$ and the set of predecessors of C_k in G' is included in $\{C_1, ..., C_{i-1}\}$. Since G' is a valid plan, the set of predecessors of C_k in $\mathcal{G}$ is included in $\{C_1, ..., C_{i-1}\}$. Then, we prove that the value L_{C_k} computed at line 9 of the algorithm at step i verifies $L_{C_k} \leq L'(k)$ (see Property A below). Then $L(i) \leq L_{C_k} \leq L'(k) \leq L'(i)$.
(ii) If the set of predecessors of C_i in G' is included in $\{C_1, ..., C_{i-1}\}$, then we can prove that $L'(i) \geq L_{C_i} = L(i)$, where L_{C_i} is the value computed at step i (see Property B below).
In both cases (i) and (ii), since $L'(i)$ is optimal, we have $L(i) = L'(i)$, thus proving the optimality of $L(i)$. It remains to prove the two assumptions concerning values of L_C (properties A and B).

Proof of Properties A and B. Let C_k be a service with $k \geq i$ ($k > i$ for Property A, $k = i$ for Property B). Let G' be a valid plan such that the set of predecessors of C_k is included in $\{C_1, ..., C_{i-1}\}$.

Then we prove that $L'(k) \geq L_{C_k}$, where L_{C_k} is the value computed at step i of the algorithm. Let $S = \{C_{u_1}, ..., C_{u_l}\}$ be the set of predecessors of C_k in G'. Let S' be the set of services that are either in S, or predecessor of a service of S in G. Let us show that $\prod_{C_i \in S} \sigma_i \geq \prod_{C_i \in S'} \sigma_i$. Let S_1 be the set of predecessors of C_{u_1} in G, S_2 the set of predecessors of C_{u_2} in G not in $S_1 \cup \{C_{u_1}\}$ and for all i S_i the set of predecessors of C_{u_i} in G not in $\bigcup_{j<i} S_j \cup \{C_{u_{i_1}}, ..., C_{u_{i-1}}\}$. Suppose that for one of the sets S_i, the product of selectivities $\prod_{C_j \in S_i} \sigma_j$ is strictly greater than one. Then $S_1 \cup ... \cup S_{i-1} \cup \{C_{u_{i_1}}, ..., C_{u_{i-1}}\}$ is a valid subset for C_{u_i} because G' is a valid plan and the product of selectivities on this subset is strictly smaller than the product of selectivities of the predecessors of C_{u_i} in G. This is in contradiction with the optimality of the set of predecessors of C_{u_i} chosen at line 7 of the algorithm. This proves that for all i, $\prod_{C_j \in S_i} \sigma_j \leq 1$. In addition, for all $j < i$, $L(j)$ is optimal. Hence the latency of C_k in G with S' as predecessor is smaller or equal to its latency in G', which proves that $L'(k) \geq L_{C_k}$.

Thus, for $1 \leq i \leq n$, $L(i)$ is optimal, and therefore the plan computed by Algorithm 2 is optimal. ■

Algorithm 3: Optimal algorithm for BICRITERIA-INDEP-HOM.

Data: n services with selectivities $\sigma_1, ..., \sigma_p \leq 1, \sigma_{p+1}, ..., \sigma_n > 1$, ordered costs $w_1 \leq \cdots \leq w_p$, and a maximum period K
Result: a plan G optimizing the latency with a period less than K

```
1  G is the graph reduced to node C_1;
2  if w_1 > K then
3  |  return false;
4  end
5  for i = 2 to n do
6  |  for j = 0 to i − 1 do
7  |  |  Compute the completion time t_j of C_i in G with predecessors C_1, ..., C_j;
8  |  end
9  |  Let S = {k | w_i ∏_{1≤l≤k} σ_l ≤ K};
10 |  if S = ∅ then
11 |  |  return false;
12 |  end
13 |  Choose j such that t_j = min_{k∈S}{t_k};
14 |  Add the node w_i and the edges C_1 → C_i, ..., C_j → C_i to G;
15 end
```

Bi-criteria problem

In the following, we prove the polynomial complexity of problems BICRITERIA-INDEP-HOM and BICRITERIA-PREC-HOM.

The following algorithms use the same principles that Algorithms 1 and 2 with adding a constraint of the execution time of services.

Theorem 2.3. *Problem* BICRITERIA-INDEP-HOM *is polynomial and of complexity at most* $O(n^2)$. *Problem* BICRITERIA-PREC-HOM *is polynomial and of complexity at most* $O(n^6)$.

Proposition 2.1. *Algorithm 3 computes the optimal latency for a bounded period with independent services (problem* BICRITERIA-INDEP-HOM*).*

Algorithm 4: Optimal algorithm for BICRITERIA-PREC-HOM.

Data: n services, a set $\mathcal{G}$ of dependence constraints and a maximum period K
Result: a plan G optimizing the latency

G is the graph reduced to the node C of minimal cost with no predecessor in $\mathcal{G}$;
if $c > K$ **then**
  return false;
end
for $i = 2$ **to** n **do**
  // At each step we add one service to G, hence the $n-1$ steps;
  Let S be the set of services not yet in G and such that their set of predecessors in $\mathcal{G}$ is included in G;
  for $C_j \in S$ **do**
    for $C_k \in G$ **do**
      Compute the set S' minimizing the product of selectivities among services of latency less than $L(k)$, and including all predecessors of C_j in $\mathcal{G}$ (using an algorithm from [16], whose execution time is $O(n^3)$);
    end
    Let S_j be the set that minimizes the latency of C_j in G with a period bounded by K, L_j be this latency and P_j be the computation time of C with the set of predecessors S_j;
  end
  if $\{C_k, C_k \in S \text{ and } P_j \leq K\} = \emptyset$ **then**
    return false;
  end
  Choose a service C_j such that $L_C = \min\{L_{C_k}, C_k \in S \text{ and } P_j \leq K\}$;
  Add to G the node C_j, and $\forall C_k \in S_j$, the edge $C_k \rightarrow C_j$;
end

Proof. The proof is similar to that of Theorem 2.1. We restrain the choice of services that can be assigned: we can only consider those whose cost, taking the combined selectivity of their predecessors into account, is small enough to obtain a computation time smaller than or equal to K. If there is no choice for a service, then it will be impossible to assign the next services either, and there is no solution. ■

Proposition 2.2. *Algorithm 4 computes the optimal latency for a bounded period (problem* BICRITERIA-PREC-HOM*).*

Proof. The proof is similar to that of Theorem 2.2. We restrain the choice of sets that can be assigned as set of predecessors: we can only consider those whose product of selectivities is small enough to obtain a computation time smaller than or equal to K for the service considered. If there is no possible set for every possible services, then the bound on the period cannot be respected. ■

2.3.3 Heterogeneous platforms

We investigate now the optimization problems with heterogeneous resources. We show that both period and latency minimization problems are NP-hard, even for independent services. Thus, bi-criteria problems on heterogeneous platforms are NP-hard. We also prove that there exists no approximation algorithm for MINPERIOD-INDEP-HET with a constant factor, unless P=NP.

Period

In the following, we show that the problem MINPERIOD-INDEP-HET is NP-complete. The following property was presented in [71] for homogeneous platforms, and we extend it to different speed servers.

Proposition 2.3. *Let $(\mathcal{F}, \mathcal{S})$ be an instance of the problem* MINPERIOD-INDEP-HET. *We suppose that $\sigma_1, \sigma_2, \cdots, \sigma_p < 1$ and $\sigma_{p+1}, \cdots, \sigma_n \geq 1$. Then the optimal period is obtained with a plan as in Figure 2.4.*

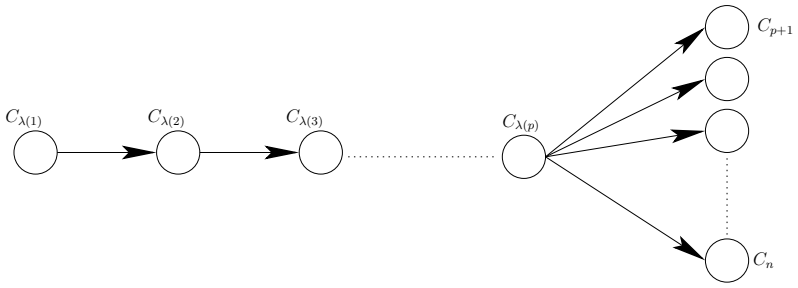

Figure 2.4: General structure for period minimization.

Proof. Let G be an optimal plan for this instance. We will not change the allocation of services to servers. Hence, in the following, C_i denotes the pair (C_i, S_u), with S_u the server assigned to C_i in G. Let $i, j \leq p$ (recall that p is the largest index of services whose selectivity is smaller than 1). Suppose that C_i is not an ancestor of C_j and that C_j is not an ancestor of C_i. Let $A'_k(G) = \mathsf{Ancest}_k(G) \cap \{C_1, ..., C_p\}$. Informally, the idea is to add the arc (C_i, C_j) to G and to update the list of ancestors of each node (in particular, removing all nodes whose selectivity is greater than or equal to 1). Specifically, we construct the graph G' such that:

- for every $k \leq p$ such that $C_i \notin \mathsf{Ancest}_k(G)$ and $C_j \notin \mathsf{Ancest}_k(G)$, $\mathsf{Ancest}_k(G') = A'_k(G)$;
- for every $k \leq p$ such that $C_i \in \mathsf{Ancest}_k(G)$ or $C_j \in \mathsf{Ancest}_k(G)$, $\mathsf{Ancest}_k(G') = A'_k(G) \cup A'_i(G) \cup A'_j(G) \cup \{C_i, C_j\}$;
- $\mathsf{Ancest}_i(G') = A'_i(G)$;
- $\mathsf{Ancest}_j(G') = A'_j(G) \cup A'_i(G) \cup \{C_i\}$;
- for every $k > p$, $\mathsf{Ancest}_k(G') = \{C_1, ..., C_p\}$.

In G', C_i is a predecessor of C_j and for all $p < k \leq n$, C_k has no successor. Also, because C_i and C_j were not linked by a precedence relation in G, G' is always a DAG (no cycle). In addition, for every node C_k of G, we have $\mathsf{Ancest}_k(G') \supset A'_k(G) = \mathsf{Ancest}_k(G) \cap \{C_1, ..., C_p\}$. This property implies:

$$C'_{\text{exec}}(k) = \frac{w_k}{s_u} \times \prod_{C_l \in \mathsf{Ancest}_k(G')} \sigma_l \leq \frac{w_k}{s_u} \times \prod_{C_l \in A'_k(G)} \sigma_l \leq \frac{w_k}{s_u} \times \prod_{C_l \in \mathsf{Ancest}_k(G)} \sigma_l \leq C_{\text{exec}}(k).$$

Hence, $\mathcal{P}(G') \leq \mathcal{P}(G)$ (recall that $\mathcal{P}(G)$ denotes the period of G). Because G was optimal, $\mathcal{P}(G') = \mathcal{P}(G)$, and G' is optimal too. By induction we construct a plan with the structure of Figure 2.4. ■

We point out that only the *structure* of the plan is specified by Proposition 2.3. There remains to find the optimal ordering of services C_1 to C_p in the chain (this corresponds to the permutation λ in Figure 2.4), and to find the optimal assignment of services to servers.

Theorem 2.4. MINPERIOD-INDEP-HET *is NP-hard.*

Proof. Consider the decision problem associated to MINPERIOD-INDEP-HET: given an instance of the problem with n services and $p \geq n$ servers, and a bound K, is there a plan whose period does not exceed K? This problem obviously is in NP: given a bound and a mapping, it is easy to compute the period, and to check that it is valid, in polynomial time.

To establish the completeness, we use a reduction from RN3DM, a special instance of Numerical 3-Dimensional Matching that has been proved to be strongly NP-complete by Yu [85, 86]. Consider the following general instance $\mathcal{I}_1$ of RN3DM: given an integer vector $A = (A[1], \ldots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that

$$\forall 1 \leq i \leq n, \quad \lambda_1(i) + \lambda_2(i) = A[i] \tag{2.1}$$

We can suppose that $2 \leq A[i] \leq 2n$ for all i and that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution. Then we point out that Equation (2.1) is equivalent to

$$\begin{aligned} & \forall 1 \leq i \leq n, \quad \lambda_1(i) + \lambda_2(i) \geq A[i] \\ \iff \quad & \forall 1 \leq i \leq n, \quad \left(\tfrac{1}{2}\right)^{\lambda_1(i)-1} \times \tfrac{2^{A[i]}}{2^{\lambda_2(i)}} \leq 2 \end{aligned} \tag{2.2}$$

We build the following instance $\mathcal{I}_2$ of MINPERIOD-HET with n services and $p = n$ servers such that $w_i = 2^{A[i]}$, $\sigma_i = 1/2$, $s_i = 2^i$ and $K = 2$. The size of instance $\mathcal{I}_1$ is $O(n \log(n))$, because each $A[i]$ is bounded by $2n$. In fact, because RN3DM is NP-complete in the strong sense, we could encode $\mathcal{I}_1$ in unary, with a size $O(n^2)$, this does not change the analysis. We encode the instance of $\mathcal{I}_1$ with a total size $O(n^2)$, because the w_i and s_i have size at most $O(2^n)$, hence can be encoded with $O(n)$ bits each, and there are $O(n)$ of them. The size of $\mathcal{I}_2$ 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. Assume first that $\mathcal{I}_1$ has a solution. Then we build a plan which is a linear chain. Service C_i is at position $\lambda_1(i)$, hence is filtered $\lambda_1(i) - 1$ times by the previous services, and it is processed by server $S_{\lambda_2(i)}$, matching the cost in Equation (2.2).

Reciprocally, if we have a solution to $\mathcal{I}_2$, then there exists a linear chain G with period 2. Let $\lambda_1(i)$ be the position of service C_i in the chain, and let $\lambda_2(i)$ be the index of its server. Equation (2.2) is satisfied for all i, hence Equation (2.1) is also satisfied for all i: we have found a solution to $\mathcal{I}_1$. This completes the proof. ■

The proof also shows that the problem remains NP-complete when all service selectivities are identical.

Proposition 2.4. *For any $K > 0$, there exists no K-approximation algorithm for* MINPERIOD-INDEP-HET*, unless P=NP.*

Proof. Suppose that there exists a polynomial algorithm that computes a K-approximation of this problem. We use the same instance $\mathcal{I}_1$ of RN3DM as in the proof of Theorem 2.4: given an integer vector $A = (A[1], \ldots, A[n])$ of size $n \geq 2$, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that $\forall 1 \leq i \leq n, \quad \lambda_1(i) + \lambda_2(i) = A[i]$. We can suppose that $2 \leq A[i] \leq 2n$ for all i and that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution.

Let $\mathcal{I}_2$ be the instance of our problem with n services with, for $1 \leq i \leq n$, $w_i = (2K)^{A[i]-1}$, $\sigma_i = \frac{1}{2K}$, $s_i = (2K)^i$ and $P = 1$. The only optimal solutions are the chains such that the service C_i is placed in position $\lambda_1(i)$ in the chain, and it is processed by server $S_{\lambda_2(i)}$, where (λ_1, λ_2) is a solution of $\mathcal{I}_1$. In any other solution, there is a service whose computation cost is larger than $P = 1$. In addition, all computation costs are integer power of $2K$. That means that in any other solution, the period is

greater or equal to $2K$. Hence the only K-approximations are the optimal solutions. If a polynomial algorithm finds such a solution, we can compute the permutations λ_1 and λ_2 and solve $\mathcal{I}_1$ in polynomial time. This contradicts the hypothesis $P \neq NP$. ■

Latency

In the following, we first show that the optimal solution of MINLATENCY-INDEP-HET has a particular structure. We then use this result to derive the NP-completeness of the problem.

Definition 2.1. *Given a plan G and a vertex $v = (C_i, S_u)$ of G, (i) v is a leaf if it has no successor in G; and (ii) $d_i(G)$ is the maximum length (number of links) in a path from v to a leaf. If v is a leaf, then $d_i(G) = 0$.*

Proposition 2.5. *Let $C_1, ..., C_n, S_1, ..., S_n$ be an instance of* MINLATENCY. *Then, the optimal latency can be obtained with a plan G such that, for any couple of nodes of G $v_1 = (C_{i_1}, S_{u_1})$ and $v_2 = (C_{i_2}, S_{u_2})$,*

1. *if $d_{i_1}(G) = d_{i_2}(G)$, v_1 and v_2 have the same predecessors and the same successors in G;*
2. *if $d_{i_1}(G) > d_{i_2}(G)$ and $\sigma_{i_2} \leq 1$, then $w_{i_1}/s_{u_1} < w_{i_2}/s_{u_2}$;*
3. *all nodes with a service of selectivity $\sigma_i > 1$ are leaves ($d_i(G) = 0$).*

Proof. Let G be an optimal plan for this instance. We will not change the allocation of services to servers, so we can design vertices of the graph as C_i only, instead of (C_i, S_u). We want to produce a graph G' which verifies Proposition 2.5.

Property 1. In order to prove Property 1 of the proposition, we recursively transform the graph G, starting from the leaves, so that at each level every nodes have the same predecessors and successors.

For every vertex C_i of G, we recall that $d_i(G)$ is the maximum length of a path from C_i to a leaf in G. Let $A_i = \{C_j |\ d_j(G) = i\}$. A_0 is the set of the leaves of G. We denote by G_i the subgraph $A_0 \cup ... \cup A_i$. Note that these subgraphs may change at each step of the transformation, and they are always computed with the current graph G.

• *Step* 0. Let $w_i = \max_{C_j \in A_0} w_j$. Let G' be the plan obtained from G by changing the predecessors of every service in A_0 such that the predecessors of a service of A_0 in G' are exactly the predecessors of C_i in G. Let B_i be the set of predecessors of C_i in G and let $C_j \in B_i$ be the predecessor of C_i of maximal completion time. The completion time of a service C_ℓ of $G - A_0$ does not change: $L'(\ell) = L(\ell)$. And, for each service C_k in A_0,

$$\begin{aligned} L'(k) &= L'(j) + \left(\prod_{C_\ell \in B_i} \sigma_\ell\right) \times w_k \\ &\leq L'(j) + \left(\prod_{C_\ell \in B_i} \sigma_\ell\right) \times w_i \\ &\leq L'(i) = L(i) \end{aligned}$$

Therefore, $\forall C_k \in A_0,\ L'(k) \leq L(i)$. Since for $C_k \notin A_0, L'(k) \leq L(k)$, and since G was optimal for the latency, we deduce that G' is also optimal for the latency. This completes the first step of the modification of the plan G.

• *Step* i. Let i be the largest integer such that G_i verifies Property 1. If $G_i = G$, we are done since the whole graph verifies the property. Let $C_{i'}$ be a node such that $L_i(i') = \max_k L_i(k)$. Note that these finish times are computed in the subgraph G_i, and thus they do not account for the previous selectivities in the whole graph G. Let C_j be an entry node of G_i (no predecessors in G_i) in a path realizing the maximum time $L_i(i')$, and let C_ℓ be the predecessor in G of C_j of maximal finish time $L(\ell)$. Then G' is

the plan obtained from G in changing the predecessors of every service of A_i such that the predecessors of a service of A_i in G' are the predecessors of C_j in G. For $C_k \in G \setminus G_i$, we have $L'(k) = L(k)$. Let C_k be a node of G_i. We have:

$$\begin{aligned} L'(k) &= L'(\ell) + \left(\prod_{C_m \in \mathsf{Ancest}_j(G')} \sigma_m\right) \times L_i(k) \\ &\leq L(\ell) + \left(\prod_{C_m \in \mathsf{Ancest}_j(G)} \sigma_m\right) \times L_i(i') \\ &\leq L(G) \end{aligned}$$

and $L(G)$ is optimal. So, $L(G') = L(G)$.

• *Termination of the algorithm.* Let C_k be a node of G. If C_k is a predecessor of C_j in G or if $C_k \in G_i$, then $d_k(G') = d_k(G)$. Otherwise, every path from C_k to a leaf in G has been removed in G', so $d_k(G') < d_k(G)$. This proves that $\sum_j d_j(G) \geq \sum_j d_j(G')$.

- If, at the end of step i, $\sum_j d_j(G) = \sum_j d_j(G')$, then G_{i+1} verifies Property 1, and we can go to step $i+1$.

- However, if $\sum_j d_j(G) > \sum_j d_j(G')$, some leaves may appear since we have removed successors of some nodes in the graph. In this case, we start again at step 0.

The algorithm will end because at each step, either the value $\sum_j d_j(G)$ decreases strictly, or it is equal but i increases. It finishes either if there are only leaves left in the graph at a step with $i = 0$, or when we have already transformed all levels of the graph and $G_i = G$.

Property 2. Let G be an optimal graph for latency verifying Property 1. Suppose that there exists a pair (C_i, S_u) and (C_j, S_v) such that $d_i(G) > d_j(G)$, $\sigma_J \leq 1$, and $w_i/s_u > w_j/s_v$. Let G' be the graph obtained by removing all the edges beginning and ending by (C_j, S_v) and by choosing as predecessors of (C_j, S_v) the predecessors of (C_i, S_u) in G and as successors of C_j the successors of C_i in G. Since $\sigma_j \leq 1$, the cost of successors can only decrease. The other edges do not change. $L(G') \leq L(G)$ and G is optimal, so G' is optimal and Property 1 of Proposition 2.5 is verified. We can continue this operation until Property 2 is verified.

Property 3. The last property just states that all nodes of selectivity greater than 1 are leaves. In fact, if such a node C_i is not a leaf in G, we remove all edges from C_i to its successors in the new graph G', thus only potentially decreasing the finish time of its successor nodes. Indeed, a successor will be able to start earlier and it will have less data to process. ■

Lemma 2.1. *Let $C_1, ..., C_n, S_1, ..., S_n$ be an instance of* MINLATENCY-HET *such that for all i, w_i and s_i are integer power of 2 and $\sigma_i \leq \frac{1}{2}$. Then the optimal latency is obtained with a plan G such that*

1. *Proposition 2.5 is verified;*
2. *for all nodes (C_{i_1}, S_{u_1}) and (C_{i_2}, S_{u_2}) with $d_{i_1}(G) = d_{i_2}(G)$, we have $\frac{w_{i_1}}{s_{u_1}} = \frac{w_{i_2}}{s_{u_2}}$.*

Proof. Let G be a plan verifying Proposition 2.5. Suppose that there exists a distance to leaves d such that the nodes at this distance do not respect Property 2 of Lemma 2.1. Let A be the set of nodes (C_i, S_u) of maximal ratio $\frac{w_i}{s_u} = c$ with $d_i(G) = d$ and A' be the set of other nodes at distance d. Let w' be the maximal ratio $\frac{w_j}{s_v}$ of nodes $(C_j, S_v) \in A'$. Since $w' < w$ and w, w' are integer power of 2, we have $w' \leq \frac{w}{2}$.

We construct the plan G' such that:

– for any node $(C_i, S_u) \notin A$, $\mathsf{Ancest}_i(G') = \mathsf{Ancest}_i(G)$;

– for any node $(C_i, S_u) \in A$, $\mathsf{Ancest}_i(G') = \mathsf{Ancest}_i(G) \cup A'$.

The completion time of nodes of A' and of nodes of distance strictly greater than d in G does not change. Let T_d be the completion time of the service (C_k, S_v) at distance $d+1$ of maximal ratio $\frac{w_k}{s_v}$. Let (C_i, S_u) be a pair of A. Let $\sigma = \sum_{C_j \in \mathsf{Ancest}_i(G)} \sigma_j$. Then we have

$$\begin{aligned} T_i(G') &= T_d + \sigma \times c' + \sigma \times (\textstyle\sum_{C_j \in A'} \sigma_j) \times c \\ &\leq T_d + \sigma \times \tfrac{c}{2} + \sigma \times \tfrac{1}{2} \times c \\ &\leq T_d + \sigma \times c \\ &\leq T_i(G) \end{aligned}$$

This proves that the completion time of the services of A does not increase between G and G'. The completion time of services of distance smaller than d does not increase because their sets of predecessors do not change. G is a graph corresponding to Proposition 2.5, that means it obtains the optimal latency; and the latency of G' is smaller or equal to the latency of G. We can conclude that G' is optimal for latency.

We obtain by this transformation an optimal plan G' for latency with strictly fewer node pairs that do not correspond to the property of Lemma 2.1 than in G. In addition, G' respects properties of Proposition 2.5. By induction, we can obtain a graph as described in Lemma 2.1. ■

Theorem 2.5. MINLATENCY-INDEP-HET *is NP-hard.*

Proof. Consider the decision problem associated to MINLATENCY-HET: given an instance of the problem with n services and $p \geq n$ servers, and a bound K, is there a plan whose latency does not exceed K? This problem obviously is in NP: given a bound and a mapping, it is easy to compute the latency, and to check that it is valid, in polynomial time.

To establish the completeness, we use a reduction from RN3DM. Consider the following general instance $\mathcal{I}_1$ of RN3DM: given an integer vector $A = (A[1], \ldots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that $\forall 1 \leq i \leq n,\ \lambda_1(i) + \lambda_2(i) = A[i]$?

We can suppose that $2 \leq A[i] \leq 2n$ for all i and that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution. We build the following instance $\mathcal{I}_2$ of MINLATENCY-HET such that $w_i = 2^{A[i] \times n + (i-1)}$, $\sigma_i = \left(\frac{1}{2}\right)^n$, $s_i = 2^{n \times (i+1)}$, and $K = 2^n - 1$.

The size of instance $\mathcal{I}_1$ is $O(nlog(n))$, because each $A[i]$ is bounded by $2n$. The new instance $\mathcal{I}_2$ has size $O(n \times (n^2))$, since all parameters are encoded in binary. The size of $\mathcal{I}_2$ is thus 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 λ_1, λ_2. We place the services and the servers on a chain with service C_i on server $S_{\lambda_1(i)}$ in position $\lambda_2(i)$ on the chain. We obtain the latency

$$\begin{aligned} L(G) &= \textstyle\sum_i \frac{w_i}{s_{\lambda_1(i)}} * \left(\frac{1}{2^n}\right)^{\lambda_2(i)-1} \\ &= \textstyle\sum_i 2^{A[i] \times n + (i-1) - n \times (\lambda_1(i)+1) - n \times (\lambda_2(i)-1)} \\ &= \textstyle\sum_i 2^{(A[i] - \lambda_1(i) - \lambda_2(i)) \times n + (i-1)} \\ &= \textstyle\sum_{i=1}^{n} 2^{i-1} \\ &= 2^n - 1 \end{aligned}$$

This proves that if $\mathcal{I}_1$ has a solution then $\mathcal{I}_2$ has a solution.

Suppose now that $\mathcal{I}_2$ has a solution. Let G be an optimal plan that respects properties of Lemma 2.1. Let (C_{i_1}, S_{u_1}), (C_{i_2}, S_{u_2}) be two distinct nodes of G. Let a_1 and a_2 be two integers such that $\frac{w_{i_1}}{s_{u_1}} = 2^{a_1}$ and $\frac{w_{i_2}}{s_{u_2}} = 2^{a_2}$. The rest of the Euclidean division of a_1 by n is equal to $i_1 - 1$, and the rest of the Euclidean division of a_2 by n is equal to $i_2 - 1$. Since both nodes are distinct, $i_1 \neq i_2$ and we can conclude that $\frac{w_{i_1}}{s_{u_1}} \neq \frac{w_{i_2}}{s_{u_2}}$. The ratios cost/speed are all different and G verifies properties of Lemma 2.1. As a result, G is a linear chain.

Let λ_1, λ_2 be two permutations such that for all i, the service C_i is in position $\lambda_2(i)$ on the server $S_{\lambda_1(i)}$. We want to achieve a latency strictly smaller than 2^n, and thus for every node $(C_i, S_{\lambda_1}(i))$,

$$\begin{array}{rrcl} & 2^{A[i]\times n+(i-1)-n\times(\lambda_1(i)+1)-n\times(\lambda_2(i)-1)} & < & 2^n \\ \iff & 2^{(A[i]-\lambda_1(i)-\lambda_2(i))\times n+(i-1)} & < & 2^n \\ \iff & A[i]-\lambda_1(i)-\lambda_2(i) & \leq & 0 \end{array}$$

This proves that λ_1, λ_2 is a valid solution of $\mathcal{I}_1$. Thus, $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ has a solution, which concludes the proof. ■

Proposition 2.6. *For any $K > 0$, there exists no K-approximation algorithm for* MINLATENCY-INDEP-HET*, unless P=NP.*

Proof. Suppose that there exists a polynomial algorithm that computes a K-approximation of this problem. Let $\mathcal{I}_1$ be an instance of RN3DM: given an integer vector $A = (A[1], \ldots, A[n])$ of size $n \geq 2$, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that $\forall 1 \leq i \leq n,\ \lambda_1(i) + \lambda_2(i) = A[i]$?

We can suppose that $2 \leq A[i] \leq 2n$ for all i and that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution.

Let $\mathcal{I}_2$ be the instance of our problem with n services such that:

- $\forall i, w_i = (2K)^{A[i]\times n^2+(i-1)}$;
- $\forall i, \sigma_i = (\frac{1}{2K})^{n^2}$;
- $\forall i, s_i = (2K)^{n^2\times(i+1)}$;
- $P = (2K)^n - 1$.

We showed in the proof of Lemma 2.1 that any optimal solution of such an instance has the structure of a chain. The optimal solutions are chains where the service C_i is associated $S_{\lambda_1(i)}$ in position $\lambda_2(i)$, with (λ_1, λ_2) is a solution of $\mathcal{I}_1$. In any other solution, there is a service with computation cost greater or equal to $(2K)^{n^2}$, that means that the latency obtained is $L \geq (2K)^{n^2}$. If there exists an algorithm that computes in polynomial time a K-approximation of this problem, on this instance, it finds in polynomial time the optimal solution. We can compute in polynomial time λ_1 and λ_2 from this solution, and then solve $\mathcal{I}_1$. That means that we can solve in polynomial tima RN3DM. However, RN3DM is NP-complete. This contradicts the hypothesis $P \neq NP$, and concludes the proof. ■

Summary

For any problem described in this section, the theoretical complexity is proved. An optimal algorithm is provided for any polynomial problem, and for any NP-complete problem, a proof of complexity is provided, and the approximation problem is solved.

Surprisingly, the complexity only depends on the platform type. All problems on homogeneous platforms are polynomial, and all problems on heterogeneous platforms are NP-complete. Therefore, the next section, considering the scheduling problems of filtering services with communication costs, is only targeting homogeneous platforms. Indeed, all problems on heterogeneous platforms are NP-complete, even without communication cost.

2.4 Problems with communication costs on homogeneous platforms

In this section, we present complexity results on homogeneous platforms with communication costs. We first detail the informations needed to detail a schedule in this context (Section 2.4.1), the commu-

nication models (Section 2.4.2) and the constraints of these models on the operation lists, that is the list of execution dates of any communication and any computation (Section 2.4.3). An example is detailed in Section 2.4.4. Then, the complexity of computing the optimal period or latency of a given execution graph, and the complexity of finding the optimal plan is studied for the period minimization in Section 2.4.5 and for the latency in Section 2.4.6.

2.4.1 Plans

We present here the informations needed to fully describe a schedule in the context of communication modeling. As in Section 2.3, the target application $\mathcal{A}$ is a set of services linked by precedence constraints. We write $\mathcal{A} = (\mathcal{F}, \mathcal{G})$ where $\mathcal{F} = \{C_1, C_2, \ldots, C_n\}$ is the set of services and $\mathcal{G} \subset \mathcal{F} \times \mathcal{F}$ is the set of precedence constraints. The target machine is a homogeneous platform with p servers (or processors) of same speed s. All servers are connected to each other by communication links of equal bandwidth bw. The cost for transmitting a data of size δ is $\frac{\delta}{\text{bw}}$.

We have to build a *plan* $PL = (EG, OL)$, where $EG = (\mathcal{C}, \mathcal{E})$ is an execution graph that summarizes all precedence relations in the mapping, and OL is an operation list that captures the occurrence of each computation and each communication. We deal with the operation lists later, after having described the communication models. As for the execution graph $EG = (\mathcal{C}, \mathcal{E})$, the nodes in $\mathcal{C}$ are the services in $\mathcal{F}$ and input/output nodes.

For each service C_k in $\mathcal{F}$, let $\mathcal{S}_{\text{in}}(k)$ be the set of its direct predecessors in EG, and let $\mathcal{S}_{\text{out}}(k)$ be the set of its direct successors. Entry nodes are nodes C_k such that $\mathcal{S}_{\text{in}}(k) = \emptyset$; for each of them we add an input node to $\mathcal{C}$ to model input from the outside world. Similarly, for each exit node C_k in $\mathcal{C}$ (with $\mathcal{S}_{\text{out}}(k) = \emptyset$), we add an output node to $\mathcal{C}$. We define:

$$C_{\text{in}}(k) = \frac{\delta_0}{\text{bw}} \sum_{C_i \in \mathcal{S}_{\text{in}}(k)} \left(\prod_{C_j \in \mathsf{Ancest}_i(EG)} \sigma_j \right);$$

$$C_{\text{comp}}(k) = \left(\prod_{C_j \in \mathsf{Ancest}_k(EG)} \sigma_j \right) \times \frac{\delta_0 \times w_k}{s};$$

$$C_{\text{out}}(k) = \frac{\delta_0}{\text{bw}} \times |\mathcal{S}_{\text{out}}(k)| \times \left(\prod_{C_j \in \mathsf{Ancest}_k(EG)} \sigma_j \right) \times \sigma_k.$$

Here, $C_{\text{in}}(k)$ is a lower bound of the time needed to receive input data from all the predecessors of C_k. The input data from each predecessor C_i is of size $\delta_0 \times \prod_{C_j \in \mathsf{Ancest}_i(EG)} \sigma_j$, hence it requires $\frac{\delta_0}{\text{bw}} \times \prod_{C_j \in \mathsf{Ancest}_i(EG)} \sigma_j$ time units for communication from C_i. We add the communication costs from all the parents (immediate predecessors) to get the total incoming communication time $C_{\text{in}}(k)$. This lower bound may not be met because of idle times due to server synchronizations for the communications. However, we have not yet specified in which order the different communications take place. This specification requires discussion of communication models. We discuss variations of both one-port [12] and multi-port models [47] in Section 2.4.2.

The outgoing communication lower bound $C_{\text{out}}(k)$ is defined similarly, except that the outgoing communication to each (immediate) successor is of same size. Finally, $C_{\text{comp}}(k)$ is the execution time of C_k on the server, with the appropriate size factor involving the selectivities of all its ancestors. We assume that each service without successor in the execution graph performs a single output communication (this models returning the results to the outside world).

As in Section 2.3, we scale all service computation costs as $w_k \leftarrow \text{bw} \times \frac{w_k}{s}$, allowing us to set $\delta_0 = \text{bw} = s = 1$. At the end of the computation, we can scale the computed period and latency by the factor $\frac{\delta_0}{\text{bw}}$ to obtain the actual values.

2.4.2 Communication models

We present here the various communication models. We first present the general overview and then we formally state the constraints and the rules of the three models. We present only an informal description of the models. Detailed formulas are provided in Section 2.4.3.

With overlap. In the first model, we assume full overlap of communications and computations, where each server can receive, compute and send (independent) data simultaneously. This model, denoted as OVERLAP, calls for multi-port communications: many incoming (resp. outgoing) communications can take place at the same time, sharing the incoming (resp. outgoing) bandwidth, provided that the total communication capacity of the server is never exceeded. Independent computations take place in parallel to these communications. In this model, the server may operate concurrently on different consecutive data sets: while receiving input for a given data set, it can execute computations for some older data set and sends output for some even older data set. We define execution time $C_{\text{exec}}(k)$ of a service/server pair C_k as the maximum execution time of the send, receive and compute operations of its service:

$$C_{\text{exec}}(k) = \max\{C_{\text{in}}(k), C_{\text{comp}}(k), C_{\text{out}}(k)\}.$$

In the overlap model, the lower bound on the period is the maximum of the quantities $C_{\text{exec}}(k)$ over all services C_k:

$$\mathcal{P} = \max_{1 \leq k \leq n} C_{\text{exec}}(k).$$

Given an execution graph, it turns out that we can generate an order of communications and computations that achieves this lower bound in the multi-port model: see Section 2.4.3. Note that the problem of determining the optimal plan is still NP-hard, and therefore, the period minimization problem is difficult. See Section 2.4.3 for a complete list of the resource constraints that need to be satisfied for the OVERLAP model.

Without overlap. In the models without overlap, a server performs communications and computations sequentially (instead of in parallel). This is typical of an execution with single-threaded programs and (one-port) serialized communications. Despite its apparent simplicity, the model calls for two variants.

- **INORDER–** In the first variant, called INORDER, each server completely processes a data set before starting the execution of the next one; it receives incoming communications for data set number, say, i, one after the other; then it executes the computations for this data set, and then it sends the output data to all its successors, one communication after the other. Only after completing this whole set of operations can the processing of data set $i + 1$ be started (with the incoming communications).
- **OUTORDER–** In this second variant, we allow for out-of-order execution, namely starting some operation (say, an incoming communication) for data set $i + 1$ (or even $i + j$, $j \geq 2$) while still processing data set i.

From an architectural point of view, we emphasize that the INORDER and OUTORDER variants may be overly pessimistic, as modern processors are capable of some internal parallelism. However, both operation modes correspond to blocking send/receive MPI primitives [68], and servers may encounter

idle time due to the synchronizations in both models. Nevertheless, we expect less idle time for the OUTORDER model than for the INORDER model, due to the additional schedule flexibility of the former model. Both variants lead to a lower bound on the computation cost for server/service C_k:

$$C_{\text{exec}}(k) = C_{\text{in}}(k) + C_{\text{comp}}(k) + C_{\text{out}}(k).$$

As before, a lower bound on the period is the maximum of the execution times. But unlike the OVERLAP model with multi-port communications, this lower bound cannot always be reached: see the example in Section 2.4.4. Note that the multi-port model is more flexible: since it permits sending data to many other servers simultaneously, orchestrating the communications in the multi-port model is an easier task than for the one-port model. We refer to Section 2.4.3 for a list of resource constraints to be enforced for each model. We emphasize that there is no closed-form formula for the period with the INORDER and OUTORDER models, which we believe is a new and surprising observation.

Latency. We have just seen that models have a strong impact on the computation of the period. This is also true for the latency (or response time), but to a lesser extent. The overlap/no-overlap distinction is no longer meaningful for optimizing this criterion. Indeed, we can always fully serialize the processing of each data set and minimize the execution time, or makespan, when processing a unique data set. In other words, we delay the processing of the next data set until the current one is completely executed, this suppresses all resource conflicts. With such a strategy, the period is equal to the latency, which in turn is equal to longest path from an input node to an output node in the plan. However, the choice between one-port or multi-port communications does have an impact on the latency. This is illustrated by the example presented in Section 2.4.4.

Other variants. Altogether, we have three models, one multi-port model with overlap and two one-port variants without overlap; the precise constraints that need be enforced are detailed in Section 2.4.3. We note that other models can be introduced, for instance one-port communications with computation/communication overlap. However, we believe that we address the most realistic combinations: on single-threaded machines it is hard to avoid doing everything sequentially, and on multi-threaded machines, we can execute computations and (several) communications concurrently. Another possibility is to consider preemptive models where communication and/or computation can be interrupted, and the bandwidth of communication can vary during the communication. Such preemptive models are beyond the scope of this chapter.

We point out that the effective difference between one-port and multi-port communications is not obvious: in most cases, the optimal solution for the multi-port model obeys the one-port constraints. In Section 2.4.4, we present examples of plans were the optimal latency and period for the multi-port model are strictly smaller than the optimal ones for the one-port model.

Characterizing solutions. In the following, we study two optimization problems: (i) MINPERIOD-COMMMODEL: find a plan $PL = (EG, OL)$ that minimizes the period; and (ii) MINLATENCY-COMMMODEL: find a plan $PL = (EG, OL)$ that minimizes the latency with COMMMODEL the considered communication model.

For each problem instance, independently of the model and of the objective function, the solution includes the execution graph EG that describes the set Ancest_i for each service C_i. But this graph alone does not give enough information to compute the schedule, i.e., the moment at which each operation takes place. We also need the complete list of the time-steps at which every communication or computation begins and ends. In this chapter, we only consider cyclic schedules, that is, schedules that repeat

for each data set. Therefore, the description of the operation list is polynomial (actually, quadratic) in the number of services.

Formally, we define the operation list OL as follows:

- For each service C_i, $\mathsf{BeginCalc}^n_{(i)}$ is the time-step where the computation of C_i on data set number n begins, and $\mathsf{EndCalc}^n_{(i)}$ is the time-step where this computation ends.
- For each edge $C_i \to C_j$ in the plan, $\mathsf{BeginComm}^n_{(i,j)}$ is the time-step where this communication involving data set number n begins, and $\mathsf{EndComm}^n_{(i,j)}$ is the time-step where this communication ends.
- The schedule starts at time-step 0 with the data set number 0, and we impose a cyclic behavior of period λ:

$$\begin{cases} \mathsf{BeginCalc}^n_{(i)} = \mathsf{BeginCalc}^0_{(i)} + \lambda \times n & \text{for each service } C_i; \\ \mathsf{EndCalc}^n_{(i)} = \mathsf{EndCalc}^0_{(i)} + \lambda \times n & \text{for each service } C_i; \\ \mathsf{BeginComm}^n_{(i,j)} = \mathsf{BeginComm}^0_{(i,j)} + \lambda \times n & \text{for each communication } C_i \to C_j; \\ \mathsf{EndComm}^n_{(i,j)} = \mathsf{EndComm}^0_{(i,j)} + \lambda \times n & \text{for each communication } C_i \to C_j. \end{cases}$$

For every model, we have rules that must be satisfied by the operation list in order to have a valid schedule. These rules ensure that no resource constraint or model assumption is violated. For instance, in the INORDER model, $\mathsf{EndComm}(j,k)^n < \mathsf{BeginComm}(i,j)^{n+1}$ for all services i, j, k and all data sets, since all work for one data set must be done before starting work on another data set. See Section 2.4.3 for the complete list of rules.

Note that all models are non-preemptive: once initiated, a communication or a communication cannot be interrupted. Also, communications are synchronous, and the bandwidth assigned to a given communication remains the same during its whole execution (this is not really a restriction for the one-port model but it is an important one for the multi-port model). With the operation list we can define the period and the latency of a plan PL:

- the period is $\mathcal{P} = \lambda$;
- the latency is $\mathcal{L} = \max\{\mathsf{EndComm}^0_{(i,j)} | C_i \to C_j \in \mathcal{E}\}$.

Remember that output nodes execute a communication to the outside world, so that the longest path for data set number 0 ends by one such communication.

2.4.3 Operation lists

Given a plan $PL = (EG, OL)$, the operation list OL defines the beginning and completion times of the computation of each service C_i (values $\mathsf{BeginCalc}^0_{(i)}$ and $\mathsf{EndCalc}^0_{(i)}$ for data set 0), and of each communication $C_i \to C_j$ for each edge in the plan (values $\mathsf{BeginComm}^0_{(i,j)}$ and $\mathsf{EndComm}^0_{(i,j)}$ for data set 0). The whole operation is cyclic and repeats every λ time-steps for a new data set.

Let B_i (resp. E_i) be the remainder of the Euclidian division of $\mathsf{BeginCalc}^0_{(i)}$ (resp. $\mathsf{EndCalc}^0_{(i)}$) by λ. Similarly, let $B_{(i,j)}$ (resp. $E_{(i,j)}$) be the remainder of the Euclidian division of $\mathsf{BeginComm}^0_{(i,j)}$ (resp. $\mathsf{EndComm}^0_{(i,j)}$) by λ.

Let $PL = (EG, OL)$ be a plan for an instance $\mathcal{A} = (\mathcal{F}, \mathcal{G})$.
Recall that $C_{\text{comp}}(i) = \left(\prod_{C_k \in \mathsf{Ancest}_i(EG)} \sigma_k\right) w_i$ is the computation cost for C_i in the execution graph.
Let $\delta(i,j) = \left(\prod_{C_k \in \mathsf{Ancest}_i(EG)} \sigma_k\right)$ be the cost of the communication from C_i to C_j whenever it exists.
For consistency, note that $C_{\text{in}}(j) = \sum_{C_i \in \mathcal{S}_{\text{in}}(j)} \delta(i,j)$.

All models are non-preemptive: once initiated, a communication or a communication cannot be interrupted. Also, communications are synchronous, and the bandwidth assigned to a given communi-

cation remains the same during its whole execution (this is not really a restriction for the one-port model but it is an important one for the multi-port model).

One-port without overlap. A valid operation list for the models INORDER and OUTORDER should respect the following constraints:

- For each node C_i, $\mathsf{EndCalc}^0_{(i)} = \mathsf{BeginCalc}^0_{(i)} + C_{\text{comp}}(i)$ (computation time).
- For each edge $C_i \to C_j$, $\mathsf{EndComm}^0_{(i,j)} = \mathsf{BeginComm}^0_{(i,j)} + \delta(i,j)$ (communication time).
- For each node C_i, for each edge pair $C_j \to C_i$ and $C_k \to C_i$,
  - $\mathsf{EndComm}^0_{(j,i)} \leq \mathsf{BeginComm}^0_{(k,i)}$ or
  - $\mathsf{EndComm}^0_{(k,i)} \leq \mathsf{BeginComm}^0_{(j,i)}$.

  This is the one-port constraint: for any service, two incoming communications for a same data set do not occur at the same time.
- For each node C_i, for each edge pair $C_i \to C_j$ and $C_i \to C_k$,
  - $\mathsf{EndComm}^0_{(i,j)} \leq \mathsf{BeginComm}^0_{(i,k)}$ or
  - $\mathsf{EndComm}^0_{(i,k)} \leq \mathsf{BeginComm}^0_{(i,j)}$.

  This is the counterpart for outgoing communications.
- For each node C_i, for each edge $C_j \to C_i$, $\mathsf{EndComm}^0_{(j,i)} \leq \mathsf{BeginCalc}^0_{(i)}$.
  For any service, all incoming communications for a given data set are completed before the beginning of the computation.
- For each node C_i, for each edge $C_i \to C_j$, $\mathsf{EndCalc}^0_{(i)} \leq \mathsf{BeginComm}^0_{(i,j)}$.
  For any service, the computation is completed before the beginning of outgoing communications.

For the INORDER model, we add the following constraint: for each node i, for each edge pair $C_j \to C_i$ and $C_i \to C_k$,

$$\mathsf{EndComm}^0_{(i,k)} \leq \mathsf{BeginComm}^1_{(j,i)} = \mathsf{BeginComm}^0_{(j,i)} + \lambda \tag{2.3}$$

Constraint (2.3) states that outgoing communications for a data set are completed before the beginning of incoming communications for the next data set.

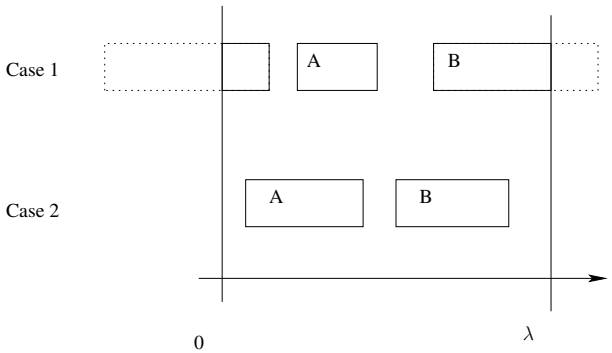

Figure 2.5: Cases for the OUTORDER model.

For the OUTORDER model, things get more complicated. We replace constraint (2.3) by the following set of constraints (see Figure 2.5):

- We forbid that an incoming communication and a computation take place at same time:
  for each edge $C_i \to C_j$,
  - $B_{(i,j)} \leq E_{(i,j)} \leq B_j \leq E_j$ (case 2, $A = C_i \to C_j$ and $B = C_j$) or

  - $B_j \leq E_j \leq B_{(i,j)} \leq E_{(i,j)}$ (case 2: $A = C_j$ and $B = C_i \rightarrow C_j$) or
  - $E_{(i,j)} \leq B_j \leq E_j \leq B_{(i,j)}$ (case 1: $A = C_j$ and $B = C_i \rightarrow C_j$) or
  - $E_j \leq B_{(i,j)} \leq E_{(i,j)} \leq B_j$ (case 1: $A = C_i \rightarrow C_j$ and $B = C_j$).
- We forbid that an outgoing communication and a computation take place at same time: for each edge $C_i \rightarrow C_j$,
  - $B_{(i,j)} \leq E_{(i,j)} \leq B_i \leq E_i$ (case 2: $A = C_i \rightarrow C_j$ and $B = C_i$) or
  - $B_i \leq E_i \leq B_{(i,j)} \leq E_{(i,j)}$ (case 2: $A = C_i$ and $B = C_i \rightarrow C_j$) or
  - $E_{(i,j)} \leq B_i \leq E_i \leq B_{(i,j)}$ (case 1: $A = C_i$ and $B = C_i \rightarrow C_j$) or
  - $E_i \leq B_{(i,j)} \leq E_{(i,j)} \leq B_i$ (case 1: $A = C_i \rightarrow C_j$ and $B = C_i$).
- We forbid that two different outgoing communications happen at the same time: for each edge pair $C_i \rightarrow C_j$ and $C_i \rightarrow C_k$,
  - $B_{(i,j)} \leq E_{(i,j)} \leq B_{(i,k)} \leq E_{(i,k)}$ (case 2: $A = C_i \rightarrow C_j$ and $B = C_i \rightarrow C_k$) or
  - $B_{(i,k)} \leq E_{(i,k)} \leq B_{(i,j)} \leq E_{(i,j)}$ (case 2: $A = C_i \rightarrow C_k$ and $B = C_i \rightarrow C_j$) or
  - $E_{(i,j)} \leq B_{(i,k)} \leq E_{(i,k)} \leq B_{(i,j)}$ (case 1: $A = C_i \rightarrow C_k$ and $B = C_i \rightarrow C_j$) or
  - $E_{(i,k)} \leq B_{(i,j)} \leq E_{(i,j)} \leq B_{(i,k)}$ (case 1: $A = C_i \rightarrow C_j$ and $B = C_i \rightarrow C_k$).
- We forbid that an incoming and an outgoing communications happen at the same time: for each edge pair $C_i \rightarrow C_j$ and $C_k \rightarrow C_i$,
  - $B_{(i,j)} \leq E_{(i,j)} \leq B_{(k,i)} \leq E_{(k,i)}$ (case 2: $A = C_i \rightarrow C_j$ and $B = C_k \rightarrow C_i$) or
  - $B_{(k,i)} \leq E_{(k,i)} \leq B_{(i,j)} \leq E_{(i,j)}$ (case 2: $A = C_k \rightarrow C_i$ and $B = C_i \rightarrow C_j$) or
  - $E_{(i,j)} \leq B_{(k,i)} \leq E_{(k,i)} \leq B_{(i,j)}$ (case 1: $A = C_k \rightarrow C_i$ and $B = C_i \rightarrow C_j$) or
  - $E_{(k,i)} \leq B_{(i,j)} \leq E_{(i,j)} \leq B_{(k,i)}$ (case 1: $A = C_i \rightarrow C_j$ and $B = C_k \rightarrow C_i$).
- We forbid that two different incoming communications happen at the same time: for each edge pair $C_j \rightarrow C_i$ and $C_k \rightarrow C_i$,
  - $B_{(j,i)} \leq E_{(j,i)} \leq B_{(k,i)} \leq E_{(k,i)}$ (case 2: $A = C_j \rightarrow C_i$ and $B = C_k \rightarrow C_i$) or
  - $B_{(k,i)} \leq E_{(k,i)} \leq B_{(j,i)} \leq E_{(j,i)}$ (case 2: $A = C_k \rightarrow C_i$ and $B = C_j \rightarrow C_i$) or
  - $E_{(j,i)} \leq B_{(k,i)} \leq E_{(k,i)} \leq B_{(j,i)}$ (case 1: $A = C_k \rightarrow C_i$ and $B = C_j \rightarrow C_i$) or
  - $E_{(k,i)} \leq B_{(j,i)} \leq E_{(j,i)} \leq B_{(k,i)}$ (case 1: $A = C_j \rightarrow C_i$ and $B = C_k \rightarrow C_i$).

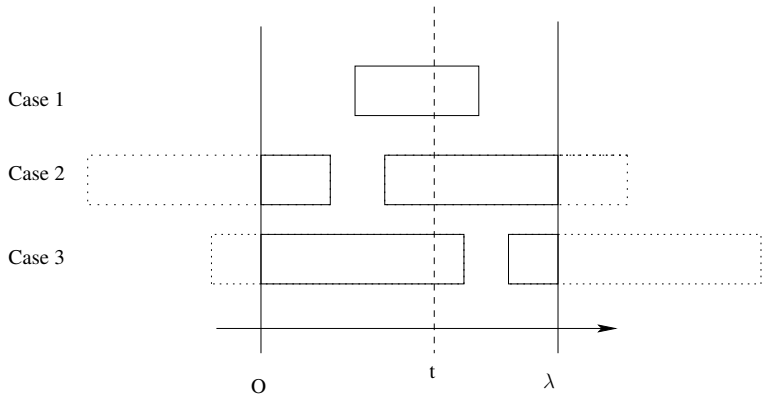

Figure 2.6: Cases for the OVERLAP model.

Multi-port with overlap. For the OVERLAP model, the servers can execute many incoming (outgoing) communications simultaneously. Bandwidth is shared between concurrent communications. Any communication on a server is assigned some ratio of the available bandwidth. This ratio does not change during the communication, hence the execution time of the communication is the cost of the commu-

nication multiplied by its bandwidth ratio. At any time, the sum of the ratios used should not exceed bw $= 1$.

We define the set of incoming communications to C_i, that are beginning or in progress at time $t + k \times \lambda$ for k large enough (see Figure 2.6): $\mathcal{A}^i_{\text{in}}(t) = \{j \in \mathcal{S}_{\text{in}}(i) | B_{(j,i)} \leq t < E_{(j,i)}$ (case 1) or $E_{(j,i)} \leq B_{(j,i)} \leq t(2)$ (case 2) or $t < E_{(j,i)} \leq B_{(j,i)}$ (case 3)$\}$.

Similarly for outgoing communications from C_i at time $t + k \times \lambda$, $\mathcal{A}^i_{\text{out}}(t) = \{j \in \mathcal{S}_{\text{out}}(i) | B_{(i,j)} \leq t < E_{(i,j)}$ (case 1) or $E_{(i,j)} \leq B_{(i,j)} \leq t(2)$ (case 2) or $t < E_{(i,j)} \leq B_{(i,j)}$ (case 3)$\}$.

The operation list is valid if:

– For all node i, $\mathsf{EndCalc}^0_{(i)} = \mathsf{BeginCalc}^0_{(i)} + C_{\text{comp}}(i)$ (computation cost).

– For each node C_i and for each edge $C_j \to C_i$,

$$\sum_{k \in \mathcal{A}^i_{\text{in}}(B_{(j,i)})} \frac{\delta(k,i)}{\mathsf{EndComm}^0_{(k,i)} - \mathsf{BeginComm}^0_{(k,i)}} \leq 1$$

(incoming communications do not exceed the bandwidth);

$$\sum_{k \in \mathcal{A}^i_{\text{in}}(B_{(j,i)})} \frac{\delta(j,k)}{\mathsf{EndComm}^0_{(j,k)} - \mathsf{BeginComm}^0_{(j,k)}}$$

(outgoing communications do not exceed the bandwidth).

– For each edge $C_i \to C_j$,

$$\mathsf{EndComm}^0_{(i,j)} \leq \mathsf{BeginCalc}^0_{(j)} \text{ and } \mathsf{EndCalc}^0_{(i)} \leq \mathsf{BeginComm}_{(i,j)}$$

(for a given data set, incoming communications are completed before the computation, which itself is completed before outgoing communications).

2.4.4 Illustrative Example

We work out the example presented in the introduction to better understand the three models. Consider an instance with 5 services, all of which have cost 4 and selectivity 1, without dependence constraints. Let the execution graph EG be the graph presented in Figure 2.3.

Latency. We start with the latency because it is simpler. Assume first one-port communications, hence the INORDER or OUTORDER models. As mentioned earlier, there is no difference between these models for computing the latency; in both cases we have to minimize the length of the longest path in the graph. If the first data set enters the graph at time $t = 0$, then the computation of service C_1 is completed at time 5. Then the computation of C_2 begins at time 6 if C_1 sends to C_2 at time 5 before sending to C_4 at time 6. The computation of C_3 begins at time 11. The computation of C_4 begins at time 7 and completes at time 11. Then, the communication between C_4 and C_5 can be done at time 12. In the meantime, C_3 completes its computation at time 15. Then, the computation of C_5 can begin at time 16 and is completed at time 20. With the last communication of C_5, this leads us to a latency of 21, which is the optimal value for the one-port model.

This execution scheme is presented in Figure 2.7. With multi-port communications we cannot achieve a better latency for this example, so we derive the same solution.

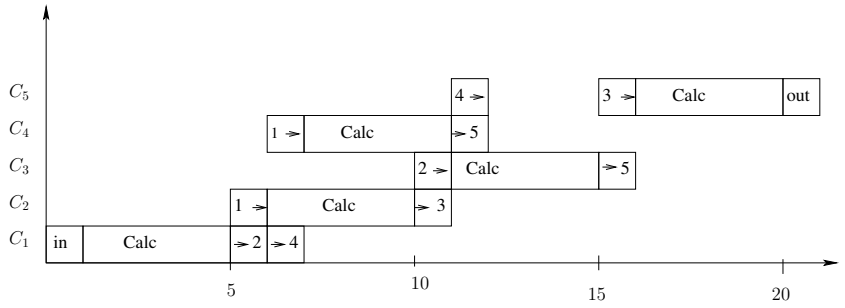

Figure 2.7: Optimal execution scheme for the latency.

Period. Looking at the above operation list, we can obtain a period $\mathcal{P} = 5$ for the OVERLAP model: if we keep the same list and only change $\lambda = 21$ into $\lambda = 5$, we have no resource conflict. In fact we can achieve a period of 4 for the OVERLAP model, and this is clearly optimal as each computation has cost 4. To do so, we modify the following in the operation list: $\lambda = 4$, $\mathsf{BeginComm}^0_{(4,5)} = 12$, and $\mathsf{EndComm}^0_{(4,5)} = 13$. For example, between time 5 and 9, server C_1 receives data set number 3, computes the data set number 2, and sends data set number 1 to C_2 and C_4. The resulting execution scheme is presented in Figure 2.8.

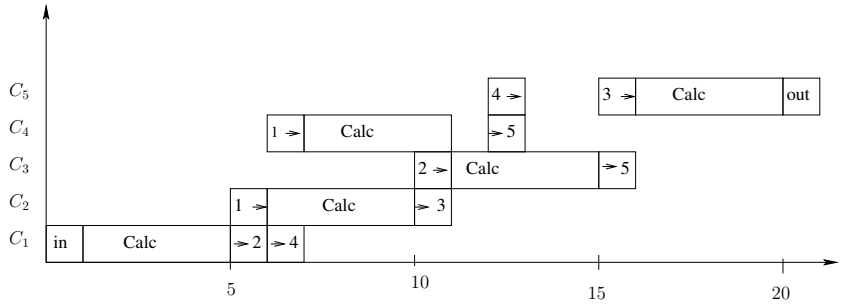

Figure 2.8: Optimal execution scheme for the period with the OVERLAP model.

For the OUTORDER model, the minimum possible period is 7, since server C_5 has two incoming communications of length 1, one computation of length 4 and one outgoing communication of length 1 (we get the same bound with C_1). This value cannot be obtained for service C_5 with the current operation list: the receive of data from C_4 for data set 1 (at time $12 + 7 = 19$) coincides with its computation for data set 0. In order to achieve a period 7, we must move the idle time to "less loaded servers." For example, we can set $\mathsf{BeginComm}^0_{(4,5)} = 14$, and $\mathsf{BeginCalc}^0_{(4)} = 8$. We keep $\mathsf{BeginComm}^0_{(1,4)} = 6$, so that there is an idle time between the end of this communication and the beginning of the computation. C_4 has another idle time at the end of this computation at time 12, and the cycle resumes for data set 1 at time $13 = 6 + 7 = \mathsf{BeginComm}^1_{(1,4)}$. The resulting execution scheme is presented in Figure 2.9.

For the INORDER model, we have the same lower bound for the period as for the OUTORDER model, namely 7. With the previous operation list, we obtain a period 10 because of the cost of C_5: the beginning of the receive for data set 1 has to wait for the end of the send of data set 0. This difference of 3 between 7 and 10 corresponds to the idle time between the end of the receive from C_4 and the beginning of the receive from C_3, which is the difference of the lengths of the path $C_1 \to C_4 \to C_5$ and of the path $C_1 \to C_2 \to C_3 \to C_5$. This idle time can be reduced by sharing it between C_1, C_4 and

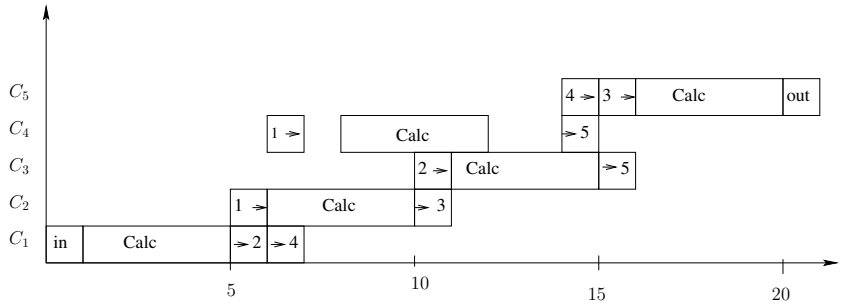

Figure 2.9: Optimal execution scheme for the period with the OUTORDER model.

C_5 as follows. The time spent in computations and communications is 7 for C_1, 6 for C_4 and 7 for C_5 respectively. The optimal solution is to give an idle time $\frac{2}{3}$ for C_1, $1+\frac{2}{3}$ for C_4 and $\frac{2}{3}$ for C_5. We obtain the following values: $\mathsf{BeginComm}^0_{(1,4)} = 6+\frac{2}{3}$,$\mathsf{EndComm}^0_{(1,4)} = 7+\frac{2}{3}$, $\mathsf{BeginCalc}^0_{(4)} = 7+\frac{2}{3}$, $\mathsf{EndCalc}^0_{(4)} = 11+\frac{2}{3}$, $\mathsf{BeginComm}^0_{(4,5)} = 13+\frac{1}{3}$, and $\mathsf{EndComm}^0_{(4,5)} = 14+\frac{1}{3}$. The other values do not change. We obtain a period $\frac{23}{3}$, which the reader may find surprising! The resulting execution scheme is presented in Figure 2.10.

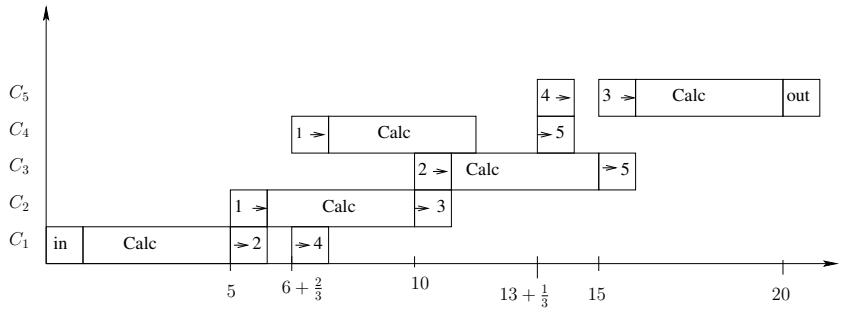

Figure 2.10: Optimal execution scheme for the period with the INORDER model.

In this example, with the same operation list, we obtain three different periods for the three different models. More interestingly, the optimal period is different for each model, and is obtained with a different operation list.

2.4.5 Period minimization

In this section, we study two problems related to period computation and minimization. First we address the following problem: given an execution graph, what is the complexity of determining the operation list that leads to the best period? We provide a polynomial algorithm for the OVERLAP model, and show that the problem is NP-hard for the INORDER and OUTORDER models. Then we address the general optimization problems MINPERIOD-OVERLAP, MINPERIOD-INORDER and MINPERIOD-OUTORDER: what is the complexity of determining the plan whose period is optimal? We show that these three problems are NP-hard. We conclude this section by providing particular polynomial instances of the period optimization problems.

Optimal period for a given execution graph

Theorem 2.6. *Given an execution graph, the problem of computing the operation list that leads to the optimal period has polynomial complexity with the* OVERLAP *model but is NP-hard with the* OUTORDER *and* INORDER *models.*

The proof of Theorem 2.6 is given by Propositions 2.7, 2.8 and 2.9.

Proposition 2.7. *Given an execution graph, the problem of computing the operation list that leads to the optimal period has polynomial complexity with the* OVERLAP *model.*

Proof. Consider an execution graph EG for an application $\mathcal{A} = (\mathcal{F}, \mathcal{G})$. Let $T = \max_{1 \leq k \leq n}\{C_{\text{exec}}(k)\}$. This value is a lower bound for period, and we prove that it can be met.

For each communication of size t, we assign to this communication a fraction t/T of the available bandwidth at the sender/receiver pair. That means that all communications will execute during T time-steps. For any server, the sum of the bandwidth ratios of incoming communications does not exceed 1, by definition of T. The same holds for outgoing communications. By definition of T, the computation time of each server also fits within the period.

We have not yet specified which data sets are operated upon by the different servers. But the previous discussion shows that every server can repeat its operations every T time-units without conflict. If suffices to let the first data set traverse the execution graph greedily: each communication is performed as soon as possible, and each computation is performed as soon as all the necessary data (all incoming communication) is available. We then repeat this scheme for every data set every T time units, and we obtain an operation list of period T. ■

Proposition 2.8. *Given an execution graph, the problem of computing the operation list that leads to the optimal period is NP-hard with the* INORDER *model.*

Proof. We consider the associated decision problem and show that is NP-complete: given an application $\mathcal{A} = (\mathcal{F}, \mathcal{G})$, an execution graph EG for this application, and a bound K, does there exist an operation list for EG such that the period does not exceed K? This problem is obviously in NP: given $\mathcal{A}$, EG and an operation list, we have the period λ and check whether it does not exceed K. To establish completeness, we use a reduction from RN3DM [86]. We consider an instance $\mathcal{I}_1$ of this problem: given an integer vector $A = (A[1], \ldots, A[n])$ of size $n \geq 2$, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that:

$$\forall 1 \leq i \leq n, \quad \lambda_1(i) + \lambda_2(i) = A[i] \tag{2.4}$$

We can suppose that $2 \leq A[i] \leq 2n$ for all i and that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance $\mathcal{I}_1$ has no solution. We associate to $\mathcal{I}_1$ an instance $\mathcal{I}_2$ with $2n + 5$ independent services, all of selectivity 1, and whose costs are as follows:

- $w_1 = w_{2n+5} = n$ and $w_{2n+3} = w_{2n+4} = 2n + 1$;
- $w_{2i} = 2n + 1$ for $1 \leq i \leq n + 1$ and $w_{2i+1} = 2n + 1 - A[i]$ for $1 \leq i \leq n$;
- $\sigma_i = 1$ for $1 \leq i \leq 2n + 5$.

The execution graph is represented in Figure 2.11. Finally, we let $K = 2n + 3$. The size of $\mathcal{I}_2$ is obviously linear in the size of $\mathcal{I}_1$.

Intuitively, we note that service C_1 has many successors and C_{2n+5} many predecessors. We need the ordering of the associated communications to compute the optimal period for this execution graph. We now 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 λ_1, λ_2. We compute the following operation list for $\mathcal{I}_2$: C_1 first communicates with C_{2n+4}. Then services C_2, C_4, …, C_{2n} are fed in the ordering given by λ_1.

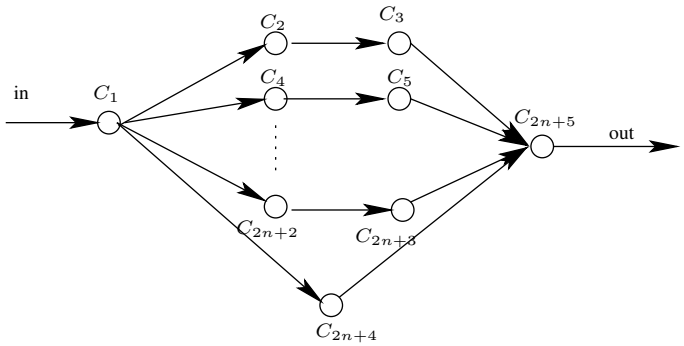

Figure 2.11: Graph G.

Finally C_{2n+4} is the last service to receive data from C_1. Receptions by C_{2n+5} are done in the order C_1, $C_{2(n-\lambda_2(1))+3}$, ..., $C_{2(n-\lambda_2(n))+3}$, C_{2n+3}. With this orchestration, owing to Equation (2.4), the period is $2n+3$.

Suppose now that $\mathcal{I}_2$ has a solution. For a data set k, suppose that the computation of C_{2n+2} begins at time i and that the computation of C_{2n+4} begins at time j. For services C_1, C_{2n+2} and C_{2n+4}, the sum of the costs of communications and of computations is equal to $2n+3$. That means that there is no idle time for the associated servers. Hence at time $i-1$ (resp. $j-1$), there is a communication between servers C_1 and C_{2n+2} (resp. C_{2n+4}) for data set k. Hence service C_1 sends the result of its computation for data set k between time-steps $i-1$ and j or between time-steps $j-1$ and i. Therefore, $|j-i|+1 \leq n+2$. For services C_{2n+3}, C_{2n+4} and C_{2n+5}, the sum of the costs of communications and of computations is equal to $2n+3$. That means that there is no idle time for the associated servers. Hence the computation of data set k on C_{2n+3} and C_{2n+4} are completed at time $i+4n+3$ and $j+2n+1$ respectively. C_{2n+5} receives the corresponding data from C_{2n+3} and C_{2n+4} at time $i+4n+3$ and $j+2n+1$ respectively. Then service C_{2n+5} receives the data for the computation of data set k between time $i+4n+1$ and $j+2n$. Hence $|(i+4n+3)-(j+2n+1)|+1 = |(i-j)+2n+2|+1 \leq n+2$. We obtain $j-i = n+1$. As a consequence, for $1 \leq i \leq n$, the communication from C_1 to C_{2i} is done between time j and $j+n$ and the communication between C_{2i+1} and C_{2n+5} is done between time $j+2n+2$ and $j+3n+2$. Let λ_1 be the ordering of communications from C_1 to services $C_2, ..., C_{2n}$ and λ_2 be the permutation such that $n+1-\lambda_2$ is the ordering of communication from $C_3, ..., C_{2n+1}$ to C_{2n+5}. We obtain

$$\begin{array}{rc} & \forall i, \lambda_1(i) + (2n+1) + 1 + (2n+1-A[i]) + \lambda_2(i) = 4n+3 \\ \iff & \forall i, \lambda_1(i) + \lambda_2(i) = A[i]. \end{array}$$

This completes the proof. ■

Proposition 2.9. *Given an execution graph, the problem of computing the operation list that leads to the optimal period is NP-hard with the* OUTORDER *model.*

Proof. Consider the associated decision problem: given an application $\mathcal{A} = (\mathcal{F}, \mathcal{G})$, an execution graph EG for this application, and a bound K, is there an operation list whose period does not exceed K? The problem is obviously in NP: given $\mathcal{A}$, EG and an operation list, we have the period λ and check whether it does not exceed K.

The NP-completeness is obtained by reduction from 2-PARTITION. Let $\mathcal{I}_1$ be an instance of 2-PARTITION that is NP-complete in the strong sense: given a set $\{a_1, ..., a_n\}$, does it exist a subset I of A such that $\sum_{a_i \in I} a_i = \frac{1}{2} \sum_{1 \leq i \leq n} a_i = A$. We construct an instance $\mathcal{I}_2$ with $2n+2$ services:

- $w_1 = An^2 - n$ and $\sigma_1 = 1$;
- $\forall 1 \leq i \leq n, w_{2i} = An^2 - 1$ and $\sigma_{2i} = An^2$;
- $\forall 1 \leq i \leq n, w_{2i+1} = 0$ and $\sigma_{2i+1} = \frac{a_i}{A}$;
- $w_{2n+2} = 0$ and $\sigma_{2n+2} = \prod_{1 \leq i \leq n} \frac{1}{a_i n^2}$;
- $K = 2An^2 + 1$;
- the plan EG is the graph presented in Figure 2.12.

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$.

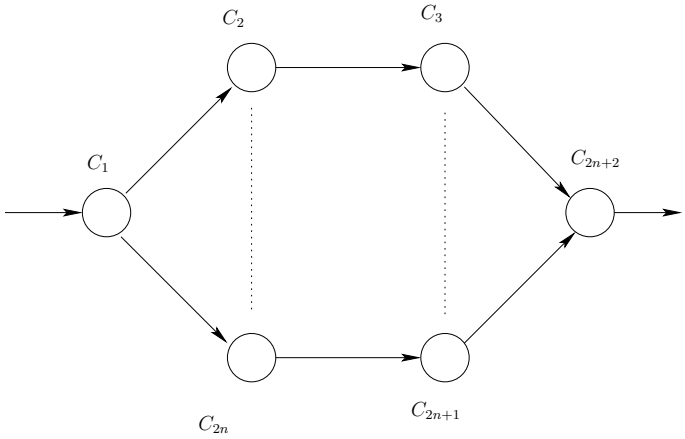

Figure 2.12: Plan EG.

Suppose that $\mathcal{I}_1$ has a solution I. Then, for any dataset k, on C_1, we first receive data, then compute dataset k, then send result, first a service C_{2i} with $a_i \in I$, then a service C_{2j}, with $a_j \notin I$, and then the remaining outgoing edges in any order. On services C_{2i} with $a_i \in I$, we receive, then compute and then send dataset k, before receiving datas for dataset $k+1$. On services C_{2i} with $a_i \notin I$, we receive dataset k, send dataset $k-1$, then compute dataset k. The service C_{2i+2} receive first datas of dataset $k-1$ from services C_{2i+1} with $a_i \in I$ in the order of emission from C_1 to C_{2i}, then comput in time 0 result of this dataset, then send result of this computation, and then receive dataset $k+1$ from C_{2i+1} with $a_i \notin I$ in order of emission from C_1 to C_{2i} with $a_i \notin I$. With the associated operation list, we obtain a latency $\lambda = K$. That means that $\mathcal{I}_2$ has a solution.

Suppose that $\mathcal{I}_2$ has a solution. Let I be the set integers a_i, $1 \leq i \leq n$ such that the execution on C_{2i} is in order, that means that for any dataset k, the emission of k to C_{2i+1} is completed before reception of dataset $k+1$ from C_1. Let t be the time of the end of the computation of C_1 for dataset k. For any value i with $a_i \in I$, the communication between C_1 and C_{2i} begin between time t and $t+n+1$ and is completed between $t+1$ and $t+n+2$. It can be no idle time on C_{2i}, that means that the communication from C_{2i+1} to C_{2i+2} begin between $t+An^2+1$ and $t+n+An^2+2$ and is completed between time $t+2An^2+1$ and $t+n+2An^2+2$. That means that the communication from C_{2i+2} to C_{2n+2} is done between time $t+2An^2+1$ and $t+n+3An^2+2$. That means that for dataset $k-1$, this communication is done between time $t+1$ and $t+An^2+n+2$. For same reasons, for any value i with $a_i \notin I$, the communication from C_{2i+2} to C_{2n+2} is done between time $t+An^2+1$ and $t+n+2An^2+2$. That means that the sum S_1 of communication costs from services $C_i, i \in I$ to C_{2n+2}, that means $\sum_{a_i \in I} a_i n^2$, is $S_1 \leq An^2+n+1$ and this sum is a multiple of n^2, that means $S_1 \leq An^2$ and the sum S_2 of communication costs from services $C_i, i \notin I$ to C_{2n+2}, that means $\sum_{a_i \notin I} a_i n^2$, is $S_2 \leq An^2+n+1$ and this sum is a multiple of n^2, that means $S_2 \leq An^2$. That means $\sum_{a_i \in I} a_i \leq A$ and $\sum_{a_i \notin I} a_i \leq A$. That proves that I is a solution to $\mathcal{I}_1$

This concludes the proof. ■

We should point out that Theorem 2.6 holds for regular streaming applications (without selectivities). This is an important and new result in that context.

Computing the optimal period

We now address the complexity of the period minimization problem for the three models. Recall that the plan consists of both the execution graph and the operation list. As it turns out, computing the optimal execution graph is NP-complete for all three period minimization problems. Therefore, even though we can compute the operation list for the OVERLAP model in polynomial time, the overall problem for computing a plan which minimizes the period is NP-complete.

On a positive note, we derived the following result on the structure of the optimal execution graph: for any instance of MINPERIOD-* without dependence constraints, and using any of the three models, there exists an optimal plan whose execution graph is a forest. This "structural" result reduces the search of optimal execution graphs. Still, all minimization problems are NP-hard.

In the optimal plan for period without communication cost, one only processor has to send data to all services of selectivity more than one. This means that in a model with communication cost, the computation time of this service is strongly increased by communications. This explains why the optimal algorithm without communication cost is no more optimal with communication costs.

Proposition 2.10. *For any instance of* MINPERIOD-* *without dependence constraints, and using any of the three models, there exists an optimal plan whose execution graph is a forest.*

Proof. For this proof, for any execution graph $EG = (\mathcal{C}, \mathcal{E})$, we define the number of added predecessors of a vertex $v \in \mathcal{C}$ as $na(v) = 0$ if v has zero or one direct predecessor, and as $na(v) = p - 1$ if v has $p \geq 2$ direct predecessors. We also define the number of added predecessors of EG as $na(EG) = \sum_{v \in \mathcal{C}} na(v)$.

Let $\mathcal{I}$ be an instance of MINPERIOD-*. Let EG be the execution graph of an optimal plan for this instance which has the minimal number of added predecessors. Suppose that EG is not a forest. Let C be a service of minimal depth with at least direct predecessors. Let C_1, C_2 be two of these predecessors.

Suppose first that C_1 and C_2 have no common predecessor. Let P_1 and P_2 the paths from an entry node to C_1 and from an entry node to C_2. There are unique by construction of C. Let Σ_1 be the product of selectivities on P_1 and Σ_2 the product of selectivities on P_2. If $\Sigma_1 \geq 1$ (resp. $\Sigma_2 \geq 1$), we can remove the edge $C_1 \to C$ (resp. $C_2 \to C$): this will decrease the product of selectivities for C as well as the output communication cost of C_1 (resp.C_2). If $\Sigma_1 < 1$ and $\Sigma_2 < 1$, let C'_2 be the root of the path P_2. C'_2 is an entry node of G by construction of P_2. If we remove the edge $C_1 \to C$ and add an edge $C_1 \to C'_2$, the product of selectivities for C does not change, and the product of selectivities for the services of P_2 decreases. These two operations strictly decrease the number of added predecessors of the graph EG and does not increase the period. This contradicts the hypothesis that EG is a optimal graph for the period with a minimal number of added predecessors.

Suppose now that C_1 and C_2 have common predecessors. Let C'_1 (resp. C'_2) be the predecessor of C_1 (resp. C_2) of minimal depth such that C'_1 (resp. C'_2) is not a predecessor of C_2 (resp. C_1). Let C' be the direct predecessor of C'_1 and C'_2. Let P_1 (resp. P_2) be the path from C'_1 (resp. C'_2) to C_1 (resp. C_2). Let Σ_1 be the product of selectivities on P_1 and Σ_2 the product of selectivities on P_2. If $\Sigma_1 \geq 1$ (resp. $\Sigma_2 \geq 1$), we can remove the edge $C_1 \to C$ (resp. $C_2 \to C$): this will decrease the product of selectivities for C as well as the output communication cost of C_1 (resp.C_2). If $\Sigma_1 < 1$, we can remove the edge $C_1 \to C$ and add an edge $C_1 \to C'_2$. The product of selectivities for C does not change, and

the product of selectivities for the services of P_2 decreases. These two operations strictly decrease the number of added predecessors of the graph EG and does not increase the period, a contradiction.

We can conclude that an optimal execution graph for the period whose number of added predecessors is minimal, necessarily is a forest. ■

Theorem 2.7. *Problems* MINPERIOD-* *without dependence constraints are all NP-hard.*

The proof of Theorem 2.7 is given by Propositions 2.11, 2.12 and 2.13.

Proposition 2.11. *The problem* MINPERIOD-OVERLAP *without dependence constraints is NP-hard.*

Proof. We consider the associated decision problem and show that is NP-complete: given n services without dependence constraints and a bound K, is there a plan whose period does not exceed K? This problem is obviously in NP: given a plan, that is an execution graph together with an operation list, we are given the period, and we can check that the operation list is valid in polynomial time. To establish the completeness, we use a reduction from RN3DM [86]. Consider an instance $\mathcal{I}_1$ of RN3DM: given an integer vector $A = (A[1], \ldots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that $\forall 1 \leq i \leq n,\ \lambda_1(i) + \lambda_2(i) = A[i]$? We can suppose that $2 \leq A[i] \leq 2n$ for all i and that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution. We associate to $\mathcal{I}_1$ an instance $\mathcal{I}_2$ of RN3DM with $3n$ services without dependence constraints. For convenience we denote these services as $C_{1,i}$, $C_{2,i}$, and $C_{3,i}$ for $1 \leq i \leq n$. We let $K = \frac{3}{2}$. Service costs and selectivities are the following:

- $\forall i$, $w_{1,i} = K$, $w_{2,i} = K \times \frac{2}{b+1}$ and $w_{3,i} = \frac{K}{a^2} \times \gamma^{-A[i]}$;
- $\forall i$, $\sigma_{1,i} = \sigma_{2,i} = a \times \gamma^i$ and $\sigma_{3,i} = \frac{K}{b^2}$.

Here, a, b and γ are rational numbers such that

- $\sqrt[2n]{\frac{3}{4}} < a < b < \sqrt[2n]{\frac{3,2}{4}}$ with $2^n \times a \in \mathbb{N}$ and $2^n \times b \in \mathbb{N}$;
- $1 < \gamma < \sqrt[n]{\frac{b}{a}}$ with $2^n \times \gamma \in \mathbb{N}$.

We have $a < b < 1$ and $\gamma \leq 2$, so their numerators are bounded by 2^{n+1}. Altogether, a, b and γ can be represented with $O(n)$ bits, which is polynomial in the size $O(n)$ of $\mathcal{I}_1$ (we have n services). But we need to prove that we can find such numbers. For a fixed n, we can find two rational numbers a and b with denominator 2^n if the function $f(n) = \sqrt[2n]{\frac{3,2}{4}} - \sqrt[2n]{\frac{3}{4}} - 2 * 2^{-n}$ is positive. We have

$$\begin{aligned} f'(n) &= -\ln(\tfrac{3,2}{4})\tfrac{1}{2n^2} \times \sqrt[2n]{\tfrac{3,2}{4}} + \ln(\tfrac{3}{4})\tfrac{1}{2n^2} \sqrt[2n]{\tfrac{3}{4}} + 2\ln(2)^{-n} \\ &\sim (\ln(\tfrac{3}{4}) - \ln(\tfrac{3,2}{4}))\tfrac{1}{2n^2} < 0 \end{aligned}$$

We obtain that f tends to 0 as n tends to $+\infty$, and that f' is negative for n big enough. This proves that there exists n_0 such that $\forall n > n_0$, $f(n) > 0$. This gives the existence of a and b for n large enough.

Now we have a and b rational numbers with denominator 2^n and both smaller than 1. Then, in worst case, $1 < \gamma < \sqrt[n]{\frac{2^n}{2^n-1}}$. Let $g(n) = \sqrt[n]{\frac{2^n}{2^n-1}} - 1 - 2^{-2n}$. Then

$$\begin{aligned} g'(x) &= \tfrac{2}{\sqrt[n]{2^n-1}}(\tfrac{\ln(2^n-1)}{n^2} - \tfrac{\ln(2)2^n}{n(2^n-1)}) + 2\ln(2)2^{-2n} \\ &= \tfrac{2}{\sqrt[n]{2^n-1}}(\tfrac{\ln(2)}{n} + \tfrac{\ln(1-2^{-n})}{n^2} - \tfrac{\ln(2)}{n(1-2^{-n})}) + 2\ln(2)2^{-2n} \\ &\sim \tfrac{2}{\sqrt[n]{2^n-1}}(\tfrac{\ln(2)}{n} - \tfrac{2^{-n}}{n^2} - \tfrac{\ln(2)}{n}(1+2^{-n})) + 2\ln(2)2^{-2n} \\ &\sim \tfrac{2}{\sqrt[n]{2^n-1}}(-\tfrac{2^{-n}}{n^2} - \tfrac{\ln(2)}{n}2^{-n}) + 2\ln(2)2^{-2n} \\ &\sim -\tfrac{2}{\sqrt[n]{2^n-1}} \times \tfrac{\ln(2)}{n}2^{-n} < 0 \end{aligned}$$

This proves the existence of γ for n big enough.

Finally, all costs and selectivities are rational numbers whose numerators and denominators are of the order at most $O(2^{n^2})$, hence the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$.

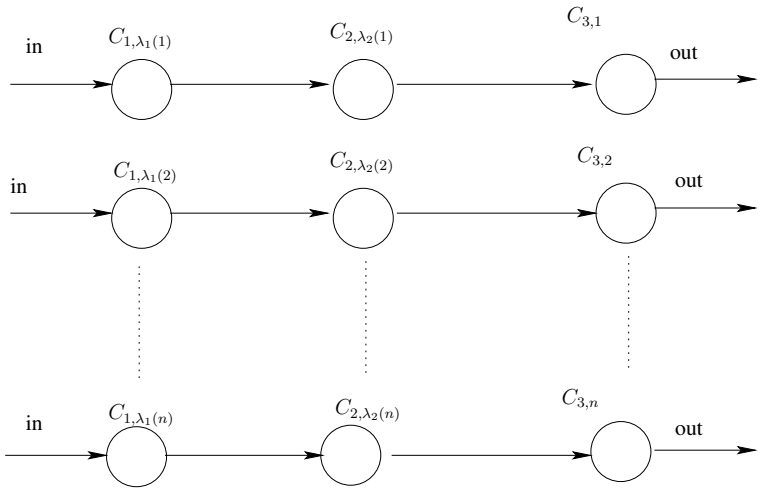

Figure 2.13: Instance $\mathcal{I}_2$.

We now 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 λ_1, λ_2. We prove that the plan whose execution graph is represented in Figure 2.13 is a solution of $\mathcal{I}_2$. For all $i, j, w_{1,i} = K$ and $\sigma_i \times C_{2,j} \leq \frac{2b}{b+1} \times K \leq K$ since $b < 1$. In addition, all communication costs are less than one and $K > 1$, then all services $C_{1,i}$ and $C_{2,i}$ respect the bound on the period. For any i, the incoming communication volume to $C_{3,i}$ is less than K. The outgoing communication volume is at most $\max_j\{\sigma_{1,j}\} \times \max_j\{\sigma_{2,j}\} \times w_{3,i} = b \times b \times \frac{K}{b^2} = K$. Concerning the computation of $C_{3,i}$, we obtain a cost

$$\begin{aligned} C_{\text{comp}}(3,i) &= \sigma_{1,\lambda_1(i)}\sigma_{2,\lambda_2(i)}w_{(3,i)} \\ &= a^2\gamma^{\lambda_1(i)+\lambda_2(i)} \times \frac{K}{a^2} \times \gamma^{-A[i]} \\ &= K \end{aligned}$$

We conclude that this plan is a valid solution of $\mathcal{I}_2$.

Suppose now that $\mathcal{I}_2$ has a solution. We prove that there exists λ_1 and λ_2 such that this solution has the plan of Figure 2.13. For all i, $w_{2,i} = K \times \frac{2}{b+1} > P$ since $b < 1$ and $w_{3,i} = \frac{K}{a^2} \times \gamma^{-A[i]} \geq \frac{K}{a^2} > K$. That means that these services cannot be entry nodes in a solution. For all i, j, $\sigma_{1,i} \times w_{3,j} \geq a \times \frac{K}{a^2}\frac{a^2}{b^2} \geq K\frac{a}{b^2} > K$ since $\frac{a}{b^2} > 1$. That means that in a solution, $C_{3,i}$ has at least 2 predecessors. Suppose that there exist i, j such that $C_{3,i}$ is predecessor of $C_{3,j}$. The outgoing communication of $C_{3,i}$ is at least: $a^{2n} \times \frac{K^2}{b^4} \geq \frac{9}{4b^2} \times K > K$. We obtain a contradiction. Suppose that there exists a service $C_{1,i}$ or $C_{2,i}$ with at least two direct successors. Then the outgoing communication of this service has a cost at least $2a^2 > \frac{3}{2} = K$. This proves that the services $C_{3,i}$ are arranged on n independent chains of length at least 3. In addition, the services $C_{2,i}$ cannot be entry nodes of the plan. This proves that the plan of the solution has the structure of Figure 2.13. Let λ_1, λ_2 be two permutations such that the predecessors of the service $C_{3,i}$ are $C_{1,\lambda_1(i)}$ and $C_{2,\lambda_2(i)}$.

The computation cost of each service $C_{3,i}$ is smaller than K:

$$\begin{array}{rll} & \forall i & C_{\text{comp}}(3,i) \leq K \\ \iff & \forall i & P \times \gamma^{\lambda_1(i)+\lambda_2(i)-A[i]} \leq K \\ \iff & \forall i & \gamma^{\lambda_1(i)+\lambda_2(i)-A[i]} \leq 1 \\ \iff & \forall i & \lambda_1(i)+\lambda_2(i)-A[i] \leq 0 \\ \iff & \forall i & \lambda_1(i)+\lambda_2(i)-A[i] = 0 \end{array}$$

Altogether, we have proven that $\mathcal{I}_1$ has a solution. This concludes the proof. ■

Proposition 2.12. *The problem* MINPERIOD-OUTORDER *without dependence constraints is NP-hard.*

Proof. We consider the associated decision problem and show that is NP-complete: given n services without dependence constraints and a bound K, is there a plan whose period does not exceed K? As before, this problem is obviously in NP. To establish the completeness, we use a reduction from RN3DM [86]. Consider an instance $\mathcal{I}_1$ of RN3DM: given an integer vector $A = (A[1], \ldots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that $\forall 1 \leq i \leq n, \quad \lambda_1(i)+\lambda_2(i) = A[i]$? We can suppose that $2 \leq A[i] \leq 2n$ for all i and that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution.

Let $x_i = y_i = n-i$ and $z_i = A[i]$ for $1 \leq i \leq n$, and let $\alpha = (1+2^{-n})$. We construct the following instance $\mathcal{I}_2$ of our problem. We have a set $\mathcal{F} = \{C_0, C_1^x, \ldots, C_n^x, C_1^y, \ldots, C_n^y, C_1^z, \ldots, C_n^z\}$ of $3n+1$ services. Each service C_i^s, where $s \in \{x, y, z\}$ has a selectivity of σ_i^s and the computation cost of w_i^s. Here is how we pick the selectivities and the computation costs for the services.

1. Let $m = 2n$. For n enough big, we have $\alpha^m < (1+\epsilon)$, where $\epsilon = 1/(2n)$.
2. Compute $K = (1+\epsilon)/(\epsilon\alpha^m)$. Note that $K = (1+\epsilon)/(\epsilon\alpha^m) > (1+\epsilon)/(1.5\epsilon) > 2n/1.5 > n+2$ for $n \geq 7$, since $\alpha^m < (1+\epsilon) < 1.5$.
3. For the first service, we set selectivity $\sigma_0 = 1/(\alpha^m(1+\epsilon))$ and $w_0 = K - 1 - n\sigma_0$. This is feasible, since $w_0 = K - 1 - n\sigma_0 > n + 2 - n + 1 = 1$. Therefore, the computation cost is always positive.
4. For services C_i^x, where $1 \leq i \leq n$, we set the selectivity such that $\sigma_i^x = \alpha^{x_i}$. Therefore, $1 < \sigma_i^x < 1+\epsilon$. We pick $w_i^x = K/\sigma_0 - \sigma_i^x - 1$. Again, the computation cost is positive.
5. For the next n services C_i^y where $1 \leq i \leq n$ we set selectivity $\sigma_i^y = (1+\epsilon)\alpha^{y_i}$ so that $1+\epsilon < \sigma_i^y < (1+\epsilon)^2$. Similarly, choose $w_i^y = K/(\sigma_0(1+\epsilon)) - 1 - \sigma_i^y$.
6. For the next n services, C_i^z for $1 \leq i \leq n$, we pick w_i^z and σ_i^z such that $1 + \sigma_i^z + w_i^z = \alpha^{z_i}K$ and set $\sigma_i^z = (1+2\epsilon)$.

The size of $\mathcal{I}_2$ is clearly polynomial in the size of $\mathcal{I}_1$.

We now 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 with permutations a and b. Then, we prove that $\mathcal{I}_2$ has a solution of the form shown in Figure 2.14. C_0 appears first, and has n outgoing links that input into services C_1^x through C_n^x. All other services just have one output. For $1 \leq i \leq n$, the output of service C_i^x goes to service $C_{\lambda_1(i)}^y$ and the output of service $C_{\lambda_1(i)}^y$ goes into the input of service $C_{\lambda_2(i)}^z$. We now prove that the period of this mapping is exactly K.

1. The period of the first service C_0 is $1+w_o+n\sigma_0 = 1+K-1-n\sigma_0+n\sigma_0 = K$, by construction.
2. The period of services C_i^x is $\sigma_0(1+w_i^x+\sigma_i^x) = \sigma_0(1+K/\sigma_0-\sigma_i^x-1+\sigma_i^x) = K$.
3. The next n services are generated using the values from set Y. The period of service $C_{\lambda_1(i)}^y$ for all $1 \leq i \leq n$ is $\sigma_0\sigma_i^x(1+w_{\lambda_1(i)}^y+\sigma_{\lambda_1(i)}^y) = \sigma_0\sigma_i^x(1+K/(\sigma_0(1+\epsilon))-1-\sigma_{\lambda_1(i)}^y+\sigma_{\lambda_1(i)}^y) = \sigma_0\sigma_i^x K/(\sigma_0(1+\epsilon)) < K$, since $\sigma_i^x < (1+\epsilon)$.

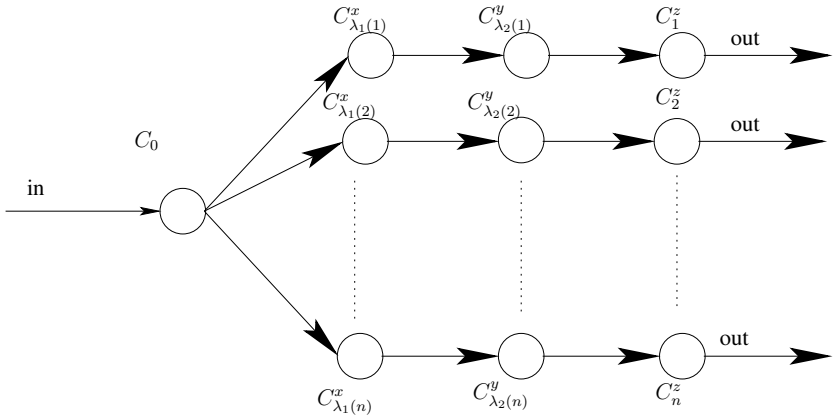

Figure 2.14: Structure of the optimal execution graph.

4. The period of service $f_{\lambda_2(i)}$ for all $1 \leq i \leq n$ is

$$\begin{aligned} P &= \sigma_0 \sigma_i^x \sigma^y_{\lambda_1(i)}(1 + w^z_{\lambda_2(i)} + \sigma^z_{\lambda_2(i)}) \\ &= \sigma_0.\alpha^{x_i}(1+\epsilon)\alpha^{y_{\lambda_1(i)}}.K\alpha^{z_{\lambda_2(i)}} \\ &= \sigma_0 \alpha^{2n}(1+\epsilon)K \\ &= \frac{1}{(1+\epsilon)\alpha^{2n}}\alpha^{2n}(1+\epsilon)K \\ &= K \end{aligned}$$

We now prove that if we have a solution to $\mathcal{I}_2$, then we have a solution to $\mathcal{I}_1$. Say we have a mapping of services with period is at most K.

Observation. *C_0 must appear first in the mapping, and all other services must come after C_0.*

Proof. If a service C^s_i, $s \in \{x, y, z\}$ is not after C_0, its period is at least $1 + w^s_i + \sigma^s_i$, since the selectivity of all other services is greater than 1.

1. For services C^x_i, the period is at least $1 + w^x_i + \sigma^x_i = K/\sigma_0 > K$, since $\sigma_0 < 1$.
2. For services C^y_i, we have period $1 + w^y_i + \sigma^y_i = K/(\sigma_0(1+\epsilon)) > K$ since $\sigma_0 = 1/((1+\epsilon)\alpha^{2n}) < 1/(1+\epsilon)$.
3. For services C^z_i, we have $1 + \sigma^z_i + w^z_i > K\alpha^{2n} > K$ by definition.

■

Observation. *By construction, we know that for the period to be less than K, C_0 can have at most n outgoing communications.*

Observation. *All of C_0's outgoing links go directly into C^x_1 through C^x_n.*

Proof. We know that $w^x_i = K/\sigma_0 - \sigma_i - 1$. Assume for the sake of contradiction that C^x_i is after some service C^s_j, $s \in \{x, y, z\}$. Then the period of C^x_i is $\sigma_0\sigma^s_j(1 + w^x_i + \sigma^x_i) = \sigma_0\sigma^s_j(K/\sigma_0) > K$ since all $\sigma^s_j > 1$. ■

Observation. *Each of these services C^x_1 through C^x_n can have at most one outgoing branch.*

Proof. If C^x_i had two branches, the period would be $\sigma_0(1 + w^x_i + 2\sigma^x_i) = \sigma_0(1 + K/\sigma_0 - 1 - \sigma^x_i + 2\sigma^x_i) = K + \sigma_0\sigma^x_i > K$. ■

Observation. *The outgoing branches from C_1^x through C_n^x go into distinct services C_1^y through C_n^y, not necessarily in order.*

Proof. Assume for contradiction that we can put service C_j^s, $s \in \{y, z\}$ between C_k^x and C_i^y. The period of C_i^y is $\sigma_0 \sigma_k^x \sigma_j^s (1 + w_i^y + \sigma_i^y) > \sigma_0(1+\epsilon)(1 + K/(\sigma_0(1+\epsilon)) - 1 - \sigma_i^y + \sigma_i^y) = K$, since $\sigma_k^x > 1$ and $\sigma_j^s > (1+\epsilon)$. ■

Observation. *Again, these services C_1^y through C_n^y can have only one outgoing edge.*

Proof. If they have 2 edges and if service C_k^x precedes this service C_i^y, their period is

$$\begin{aligned} P &= \sigma_0 \sigma_k^x (1 + w_i^y + 2\sigma_i^y) \\ &> \sigma_0 (K/(\sigma_0(1+\epsilon)) + \sigma_i^y) \\ &> K/(1+\epsilon) + \sigma_0(1+\epsilon) \\ &= K/(1+\epsilon) + K\epsilon(1+\epsilon)/(1+\epsilon)^2 \\ &= K. \end{aligned}$$

■

Observation. *Finally, all of the services C_i^z are on these outgoing n edges.*

Therefore, the structure of the graph to get a period of K is exactly as is shown in the figure.

Let us consider the service C_k^z, which is chained after services C_i^x and C_j^y. The period of service C_k^z is

$$\begin{aligned} \sigma_0 \sigma_i^x \sigma_j^y (1 + w_k^z + \sigma_k^z) &\leq K \\ \sigma_0 . \alpha^{x_i} (1+\epsilon) \alpha^{y_j} K \alpha^{z_k} &\leq K \\ \sigma_0 (1+\epsilon) K \alpha^{(x_i + y_j + z_k)} &\leq K \\ \sigma_0 (1+\epsilon) \alpha^{(x_i + y_j + z_k)} &\leq 1 \\ \frac{1}{(1+\epsilon)\alpha^{2n}} (1+\epsilon) \alpha^{(x_i + y_j + z_k)} &\leq 1 \\ \alpha^{(x_i + y_j + z_k)} &\leq \alpha^{2n} \\ x_i + y_j + z_k &\leq 2n \end{aligned}$$

Since all the sums are less than or equal to $2n$ and $\sum_{i=1}^{n} A[i] = n(n+1)$, all sums have to be equal to $2n$. Because $x_i + y_j + z_k = 2n \Leftrightarrow i + j = A[k]$, we have a solution to $\mathcal{I}_1$. This concludes the proof. ■

Proposition 2.13. *The problem* MINPERIOD-INORDER *without dependence constraints is NP-hard.*

Proof. We use the same reduction as for Proposition 2.13, because the optimal operation list fulfilled the constraints of the INORDER model. ■

Restriction to chains

For the period minimization problem with communication costs, there exists an optimal plan that is a forest. This means that the complexity of the latter problem is the same when restricting to forest-shaped execution graphs. In this section, we establish the polynomial complexity of the period minimization problem when restricting to chains. Note that in this case, the three models of communication costs are equivalent.

Proposition 2.14. *The problem* MINPERIOD-* *when restricting to linear chain execution graphs is polynomial for all models.*

Proof. On a chain of servers, the models INORDER and OUTORDER are equivalent: they lead to the same value of the period. Consider an optimal execution graph EG for either model. Let $C_i \to C_j$ be two successive services. We suppose that

$$\max\{1 + w_i + \sigma_i, \sigma_i(1 + w_j + \sigma_j)\} \leq \max\{1 + w_j + \sigma_j, \sigma_j(1 + w_i + \sigma_i)\}$$

otherwise we can exchange their positions. Let $w'_k = 1 + w_k + \sigma_k$ for all k. If $\sigma_i, \sigma_j \leq_1$, we have $w'_i \leq w'_j$, and if $\sigma_i, \sigma_j \geq 1$, then we have $\frac{\sigma_i}{w'_i} \leq \frac{\sigma_j}{w'_j}$ and the only remaining case is $\sigma_i < 1$ and $\sigma_j > 1$. Therefore the problem can be solved by the following greedy algorithm: place services of selectivity less than 1 by increasing value of w'_k, and then have them followed by services of selectivity at least 1 arranged by increasing value of $\frac{\sigma_k}{w'_k}$.

Similarly, for the model OVERLAP, we can suppose that

$$\max\{1, w_i, \sigma_i, \sigma_i w_j, \sigma_i \sigma_j\} \leq \max\{1, w_j, \sigma_j, \sigma_j w_i, \sigma_j \sigma_i\}$$

Let $w'_k = \max\{1, w_k\}$ for all k. Then we see that $\max\{w'_i, \sigma_i w'_j\} \leq \max\{w'_j, \sigma_j w'_i\}$. We obtain the same greedy algorithm as above with the new value of w'_k. ■

2.4.6 Latency minimization

This section is the counterpart of Section 2.4.5 for the latency. First we address the following problem: given an execution graph, what is the complexity of determining the operation list that leads to the best latency? This problem turns out to be NP-hard for all models (while determining the best period was polynomial for the OVERLAP model). The general optimization problems MINLATENCY-* are all NP-hard. We conclude this section by providing the complexity of the problems where we restrict the execution graph to be a chain or a forest.

Optimal latency for a given execution graph

As for the optimization of the period, the latency of a plan depends upon the operation list. We prove in this section that the computation of the optimal latency for a given execution graph is NP-hard for the three models.

Theorem 2.8. *Given an execution graph, the problem of computing the optimal operation list that leads to the optimal latency is NP-hard for the three models.*

The proof of Theorem 2.8 is given by Propositions 2.15, 2.16 and 2.17.

Proposition 2.15. *Given an execution graph, the problem of computing the optimal operation list that leads to the optimal latency is NP-hard for the model* OUTORDER.

Proof. We consider the associated decision problem: given an application $\mathcal{A} = (\mathcal{F}, \mathcal{G})$, an execution graph EG for this application, and a bound K, does it exist an operation list for EG such that the latency does not exceed K? This problem is obviously in NP: given $\mathcal{A}$, EG and an operation list, we can compute $\max\{\mathsf{EndComm}^0_{(i,j)} |\ C_i \to C_j \in \mathcal{E}\}$ and check whether this value does not exceed K.

The NP-completeness is obtained by reduction from RN3DM. Let $\mathcal{I}_1$ be an instance of RN3DM: given an integer vector $A = (A[1], \ldots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that $\forall 1 \leq i \leq n,\ \lambda_1(i) + \lambda_2(i) = A[i]$? We can suppose that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution. We associate to $\mathcal{I}_1$ an instance $\mathcal{I}_2$ with $n+2$ services C_0 to C_{n+1}, without dependence constraints, all of selectivity 1, and whose costs are as follows:

- $w_0 = w_{n+1} = 1$;
- $w_i = B[i] = n - A[i] + n^2$ for $1 \leq i \leq n$.

We let $K = n + 4 + n^2$. The execution graph is a fork-join plan EG represented in Figure 2.15. The size of the instance $\mathcal{I}_2$ is linear in the size of the instance $\mathcal{I}_1$.

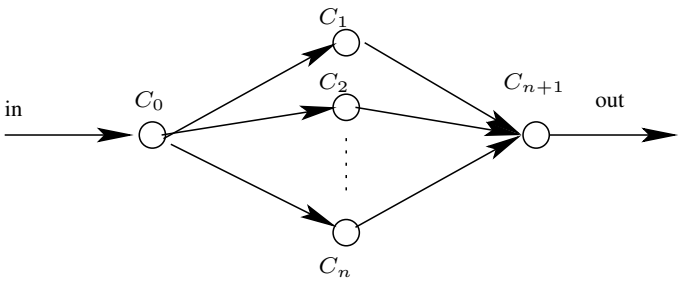

Figure 2.15: The fork-join execution graph.

We now 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 λ_1, λ_2. Then for $1 \leq i \leq n$, the services C_i are fed in the order λ_1 and the receptions are done in the order $n + 1 - \lambda_2$. For service C_i, the computation begins at time $2 + \lambda_1(i)$ and is completed at time $2 + \lambda_1(i) + B[i]$. There remain $\lambda_2(i)$ communications to do when the service C_i sends its data to service C_{n+1}. So the final latency is at least $l(i) = \lambda_1(i) + B[i] + \lambda_2(i) + 4$ for any i. In fact, we see that the latency $\mathcal{L}$ is equal to equal to $\max_i l(i)$. Hence $\mathcal{L} \leq \max_i \lambda_1(i) + B[i] + \lambda_2(i) + 4 = n + 4 + n^2$. That proves that $\mathcal{I}_2$ has a solution.

Suppose now that $\mathcal{I}_2$ has a solution. Let λ_1 be the sending order from C_0 and λ_2 be a permutation such that $n + 1 - \lambda_2$ is the order of receptions by C_{n+1}. For service C_i, the computation begins at time $2 + \lambda_1(i)$ and is completed at time $2 + \lambda_1(i) + B[i]$. There remain $\lambda_2(i)$ communications to do when the service C_i send its data to service C_{n+1}. So the final latency $\mathcal{L}$ is at least $l(i) = \lambda_1(i) + B[i] + \lambda_2(i) + 4$ for all i, and we have $\mathcal{L} \leq K = \leq n+4+n^2$. Hence $\lambda_1(i)+B[i]+\lambda_2(i) \leq n+n^2$, or $\lambda_1(i)+\lambda_2(i) \leq A[i]$ for all i. Summing up, we see that all these inequalities are in fact equalities, hence a solution to $\mathcal{I}_1$ [3]. ■

Proposition 2.16. *Given an execution graph, the problem of computing the optimal operation list that leads to the optimal latency is NP-hard for the model* INORDER.

Proof. We use the same reduction as for Proposition 2.15, because the optimal operation list fulfilled the constraints of the INORDER model. ■

3. Note that we did not define λ in $\mathcal{I}_2$. As pointed out before, we can always enforce a period λ equal to $\mathcal{L} = K$ to avoid any resource conflict.

Proposition 2.17. *Given an execution graph, the problem of computing the optimal operation list that leads to the optimal latency is NP-hard for the model* OVERLAP.

Proof. This proof uses the reduction in the proof of Proposition 2.15. Let $\mathcal{I}_1$ be an instance of RN3DM. Let $\mathcal{I}_2$ be the instance associated to $\mathcal{I}_1$ by this proof. Let EG be the execution graph presented in Figure 2.15. A valid solution for the model OUTORDER is valid for the model OVERLAP. We show the converse: for any valid solution involving multi-port communications, there is a solution involving only one-port communications and whose latency is at least as good. Intuitively, when there is a single predecessor common to several nodes, the best is to feed these nodes sequentially. Sharing the bandwidth would only delay the first communications without accelerating the other ones.

Suppose that there exists a valid operation list OL_1 for OVERLAP on instance $\mathcal{I}_2$. Let λ_1 be the completion order of the communications from C_0. We construct an operation list OL_2 such that all communications are still done in the order λ_1 but their assigned bandwidth ratios are now all equal to 1 (which means that the communications are done sequentially in OL_2). For $1 \leq i \leq n$, the communication $C_0 \to C_i$ is completed not later in OL_2 than in OL_1: if it is the $\lambda_1(i)$-th communication, it is completed at least at time $\lambda_1(i)$ after the end of the computation of C_0 in OL_1, and this value is obtained in OL_2.

Let λ_2 be the reverse order of the beginning of the communications to C_{n+1} in OL_1. In OL_2, we execute these communications in the order $n+1-\lambda_2$ with bandwidth ratio 1. The time needed for the the last i sends in OL_2 is not larger in OL_1, because it is equal to i in OL_2 and at least this value in OL_1.

In this plan, a valid operation list for the model OVERLAP is therefore valid for the model OUTORDER with the hypothesis $\lambda \geq \max\{\mathsf{EndComm}^0_{(i,j)} |\ C_i \to C_j \in \mathcal{E}\}$. This concludes the proof. ■

Algorithm 5: Computation of the latency on a tree.

Data: tree T
Result: latency L
if *T is restricted to a leaf C_i* **then**
  $L = w_i$
else
  Let $T_1, ..., T_k$ be the subtrees of the children of the root C_0 of T;
  for $i = 1$ **to** k **do** Compute the optimal latency L_i of the subgraph T_i;
  Let σ be a permutation such that $L_{\sigma(1)} \leq ... \leq L_{\sigma(k)}$;
  Let C_j be the root of T;
  Let $L = 1 + w_0 + \sigma_0 \times \max_{1 \leq i \leq k}\{(k-i+1) + L_{\sigma(i)}\}$;
end

Proposition 2.18. *Algorithm 5 computes in time $O(n \log(n))$ the optimal latency of a plan whose execution graph is a tree.*

Proof. First we note that for tree-shaped execution graph, all models are equivalent with respect to the latency: as explained in the proof of Proposition 2.17, one-port communications are always dominant.

For any node, the algorithm feeds the subtrees by non-increasing latency, which is clearly optimal. ■

As for the period (Theorem 2.6), we point out that Theorem 2.8 holds for regular streaming applications (without selectivities). Again, this is an important and new result in that context.

Computing the optimal latency

In this section, we address the complexity of the latency minimization problem for the three models. Note here that the fact that finding the operation list *given* an execution graph is NP-complete does not automatically imply that the problem of finding the plan is NP-complete. For example, the optimal plan may always consists of a simple execution graph for which the operation list can be computed in polynomial time. Therefore, in order to prove that the latency minimization problem is NP-complete, we have to argue that either (i) computing the execution graph that minimizes latency is NP-complete (as we did for the period minimization proofs) or (ii) that the plans that minimize latency contain the "difficult" execution graphs that do not allow us to compute the best operation list easily. In this instance, we prove the following result using the second option.

In Algorithm 1, we use the following property: the completion time of a service does not depend on the number of other successors of its direct predecessors. This is no more true in a model with communication cost: for a processor with many successors, we have to schedule the outgoing communications. Then, Algorithm 1 is not optimal in a model with communication costs

Theorem 2.9. *Problems* MINLATENCY-OVERLAP, MINLATENCY-OUTORDER *and* MINLATENCY-INORDER *without dependence constraints are all NP-hard.*

The proof of Theorem 2.9 is given by Propositions 2.19, 2.20 and 2.21.

Proposition 2.19. *The problem* MINLATENCY-OUTORDER *without dependence constraints is NP-hard.*

Proof. We consider the associated decision problem: given a period L, is there a mapping of latency less than L? The problem is obviously in NP: given a $\mathcal{A}$, EG and OL, we can compute $\max\{\mathsf{EndComm}^0_{(i,j)} | C_i \to C_j \in \mathcal{E}\}$ and check wether this value does not exceed K.

The NP-completeness is obtained by reduction from RN3DM, a special instance of 3-dimensional matching. Let $\mathcal{I}_1$ be an instance of RN3DM: given an integer vector $A = (A[1], \ldots, A[n])$ of size $n \geq 2$, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that $\forall 1 \leq i \leq n,\ \lambda_1(i) + \lambda_2(i) = A[i]$? We can suppose that $2 \leq A[i] \leq 2n$ for all i and that $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution. We construct an instance $\mathcal{I}_2$ with $n+2$ services as follows:

- one service F (where F stands for *fork*) with cost w_f and selectivity σ_f both equal to $\frac{1}{20n}$;
- n services C_i, $1 \leq i \leq n$, with cost $w_i = 10n - A[i]$ and selectivity $\sigma_i = \sigma = 1 - \frac{1}{2n}$;
- one service J (where J stands for *join*) with cost $w_j = 1$ and selectivity $\sigma_j = 200n^2 - 1$.

Given this construction, we ask whether this set of services can be arranged to obtain a latency at most $K = \frac{1}{2} + 10n\sigma^n + \frac{1}{20n}$. The size of $\mathcal{I}_2$ is clearly polynomial in the size of $\mathcal{I}_1$.

We now 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 with permutations λ_1 and λ_2. Then, we prove that $\mathcal{I}_2$ has a solution whose plan is a fork-join graph with F as the source, having the n services C_i as its children, and J being the final node, successor of all the n services C_i. The services C_i are fed in order λ_1 and the reception are done in order $n + 1 - \lambda_2$. For service C_i, the computation begins at $w_f + \sigma_f\lambda_1(i) = \sigma_f\lambda_1(i) + \frac{1}{20n}$ and is completed at $w_f + \sigma_f(\lambda_1(i) + w_i)$. There remain $\lambda_2(i)$ communications of size $\sigma_f\sigma \leq \sigma_f$ to execute when service C_i sends its data to final service J. The start-up time of J is bounded by $\max_i l(i)$, where $l(i) = w_f + \sigma_f(\lambda_1(i) + w_i + \sigma\lambda_2(i)) \leq w_f + \sigma_f(\lambda_1(i) + 10n - A[i] + \lambda_2(i)) = w_f + \sigma_f(10n) = \frac{1}{2} + \frac{1}{20n}$. Hence this plan achieves a latency at most $\frac{1}{2} + \frac{1}{20n} + \sigma_f\sigma^n(w_j + \sigma_j) = \frac{1}{2} + 10n\sigma^n + \frac{1}{20n} = K$. This proves that $\mathcal{I}_2$ has a solution.

We now prove that if we have a solution to $\mathcal{I}_2$, then we have a solution to $\mathcal{I}_1$. Say we have a mapping of services with latency at most K. Note that $\sigma^n < \sigma < 1$. Note also that $0.7 > \sigma^n > \frac{1}{2}$ for all n,

because the sequence $u_n = \left(1 - \frac{1}{2n}\right)^n$ is non-decreasing (and converges to $\frac{1}{\sqrt{(e)}}$, where e is the base of the natural logarithm). As a consequence, we have $K < 1/2 + 7n$.

We show that the plan necessarily is a fork-join. We make some preliminary observations:

- if one service C_i has no predecessor, then the latency is at least $w_i + \sigma \geq w_i \geq 8n > K$
- if service J has no predecessor, then the latency is at least $w_j + \sigma_j = 200n^2 > K$
- if service J is a direct successor of service F, then the latency is at least $w_f + \sigma_f(w_j + \sigma_j) \geq 10n > L$

So we know that F is a predecessor of all nodes and that the predecessors of J include F and at least one of the C_i. Assume (by contradiction) that we have exactly $k < n$ services C_i in the list of the predecessors of J. The latency obtained this way is at least L_k:

$$L_k = w_f + \sigma_f(\min(w_i) + \sigma^k(w_j + \sigma_j) \geq \frac{1}{20n} + \frac{1}{20n}\left(8n + \sigma^k 200n^2\right).$$

We derive $L_k - K \geq 10n(\sigma^k - \sigma^n) - \frac{1}{10}$. But

$$\sigma^k - \sigma^n \geq \sigma^{n-1} - \sigma^n = \sigma^{n-1}(1-\sigma) \geq \sigma^n(1-\sigma) \geq \frac{1}{2}\frac{1}{2n} = \frac{1}{4n}.$$

Hence $L_k - K \geq \frac{10n}{4n} - \frac{1}{10} > 0$, the desired contradiction, and J is a successor of all other services.

There remains to show that each service C_i is a direct successor of F to obtain the fork-join plan. But if two services C_i and C_j were serialized, the latency would be at least L', where

$$L' = w_f + \sigma_f(\min(w_i) + \sigma \min(w_i) + \sigma^n(w_j + \sigma_j).$$

We get $L' - K \geq \frac{1}{20n}8n(1+\sigma) - \frac{1}{2}$. But $\sigma > \frac{3}{4}$ since $n \geq 2$ hence $L' - K > 0$, again a contradiction.

Now that we have the fork-join plan, we assume that the services C_i are fed in order λ_1 and the reception are done in order $n + 1 - \lambda_2$. For service C_i, the latency is at least $l(i) = w_f + \sigma_f(\lambda_1(i) + w_i + \sigma\lambda_2(i))$ and we must have $l(i) \leq \frac{1}{2} + \frac{1}{20n}$ for all i. We have the following case analysis:

- If for some i we had $\lambda_1(i) + \lambda_2(i) > A[i]$, then $\lambda_1(i) + \lambda_2(i) \geq A[i] + 1$ (because we deal with integers) and we would derive $l(i) = w_f + \sigma_f(10n + \lambda_1(i) + \lambda_2(i) - A[i] + (\sigma - 1)\lambda_2(i))$ and $l(i) - \frac{1}{2} - \frac{1}{20n} \geq \sigma_f(1 - \frac{\lambda_2(i)}{2n}) > 0$, a contradiction.
- If for some i we had $\lambda_1(i) + \lambda_2(i) < A[i]$, then by symmetry we would have some i' such that $\lambda_1(i') + \lambda_2(i') > A[i']$. This is because $\sum_i \lambda_1(i) + \lambda_2(i) = n(n+1) = \sum_i A[i]$. Using i' we obtain a contradiction as above.
- We conclude that $\lambda_1(i) + \lambda_2(i) = A[i]$ for all i, hence a solution to $\mathcal{I}_1$.

This concludes the proof. ■

Proposition 2.20. *The problem* MINLATENCY-INORDER *without dependence constraints is NP-hard.*

Proof. We use the same reduction as for Proposition 2.19, because the optimal operation list fulfilled the constraints of the INORDER model.

■

Proposition 2.21. *The problem* MINLATENCY-OVERLAP *without dependence constraints is NP-hard.*

Proof. The reasoning for the proof of Proposition 2.17 can be applied to Proposition 2.19, because the optimal plans have the same structure. ■

Restriction to forests and chains

We discuss here the complexity of latency minimization problems when the mappings are restricted to forests and chains. In such cases, the latency does not depend of the communication cost model.

Proposition 2.22. *The problem* MINLATENCY-* *when restricting to plans whose execution graphs are linear chains is polynomial for all models.*

Proof. Let EG be an optimal chain for the latency. Suppose that C_i is the direct predecessor of C_j. We have

$$\begin{array}{rl} & 1 + w_i + \sigma_i + \sigma_i w_j \leq 1 + w_j + \sigma_j + \sigma_j w_i \\ \iff & \frac{1-\sigma_i}{1+w_i} \geq \frac{1-\sigma_j}{1+w_j} \end{array}$$

We obtain the following greedy algorithm : order the services by decreasing values $\frac{1-\sigma_i}{1+w_i}$. ■

Proposition 2.23. *The problem* MINLATENCY-* *when restricting to plans whose execution graphs are forests is NP-hard for all models.*

Proof. We consider the associated decision problem: given a latency K, is there a plan with an execution graph that is a forest of latency less than K? We have proved in Subsection 2.4.6 that for an execution graph that is a forest we can compute the optimal operation list for the latency in polynomial time. That means that the problem is in NP.

The NP-completeness is obtained by reduction from 2-Partition [34]. Let $\mathcal{I}_1$ be an instance from 2-Partition: given an integer set $X = \{x_1, ..., x_n\}$, does there exist a subset I such that $\sum_{x_i \in I} x_i = \frac{1}{2}\sum_{x_j \in X} x_j$? Let $x_M = \max_{x_i \in X}\{x_i\}$, $S = \sum_{x_j \in X} x_j$, $\beta = \frac{A-S}{2A+S}$ and $A > \frac{4}{3}n3^n\beta^n \times x_M^3$. We construct an instance $\mathcal{I}_2$ with $n+1$ services such that:

- $\forall i \leq n, w_i = \frac{x_i}{A}$;
- $\forall i \leq n, \sigma_i = 1 - \frac{x_i}{A} + \beta\frac{x_i^2}{A^2}$;
- $w_{n+1} = \frac{2A+S}{2A-2S}$;
- $\sigma_{n+1} = 1$;
- $K = w_{n+1} - \frac{3S^2}{8A(A-S)} + \frac{n3^n\beta^n x_M^3}{A^3}$.

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$.

We now 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 I. We place the services of I in a chain, as predecessors of C_{n+1} in any order. The remaining services are placed in parallel without any predecessor. Their latency is smaller than 1 and $K > 1$. That means that $\mathcal{I}_2$ has a solution if and only if the latency L of w_{n+1} is smaller than K. Suppose that the services in I are placed in the order $C'_1, ..., C'_{k-1}$ along the chain, and let $C_{n+1} = C'_k$. We have:

$$\begin{array}{rcl} L & = & \sum_{i<k}\prod_{j<i}\sigma'_j w'_i + \prod_{j<k}\sigma'_j w_{n+1} \\ & \leq & \sum_{i<k}\frac{x'_i}{A}(1 - \sum_{j<i}\frac{x'_j}{A} + 3^n\beta^n(\frac{x_M}{A})^2) \\ & & +w_{n+1}(1 - \sum_{i<k}\frac{x'_i}{A} + \beta\sum_{i<k}(\frac{x'_i}{A})^2 + \sum_{i<k}(\frac{x'_i}{A})^2 + 2\sum_{i<j<k}\frac{x'_i x'_j}{A^2} + 3^n\beta^n\frac{x_M^3}{A^3}) \\ & \leq & w_{n+1} + \sum_{i<k}\frac{x'_i}{A}(1 - w_{n+1}) + \sum_{i<k}(\frac{x'_i}{A})^2 w_{n+1}(\beta+1) + \sum_{i<j<k}\frac{x'_i x'_j}{A^2}(2w_{n+1} - 1) + n3^n\beta^n\frac{x_M^3}{A^3} \\ & \leq & w_{n+1} + \sum_{i<k}x'_i(\frac{-3S}{2A(A-S)}) + \sum_{i<k}x_i'^2(\frac{3}{2A(A-S)}) + \sum_{i<j<k}x'_i x'_j(\frac{1}{A(A-S)}) + n3^n\beta^n\frac{x_M^3}{A^3} \\ & \leq & w_{n+1} + (\frac{3}{2A(A-S)})(-S\sum_{i<k}x'_i + \sum_{i<k}x_i'^2 + 2\sum_{i<j<k}x'_i x'_j) + n3^n\beta^n\frac{x_M^3}{A^3} \\ & \leq & w_{n+1} + (\frac{3}{2A(A-S)})(\frac{S}{2} - \sum_{i<k}x'_i)^2 - (\frac{3}{2A(A-S)})\frac{S^2}{4} + n3^n\beta^n\frac{x_M^3}{A^3} \\ & \leq & K \end{array}$$

Then, the instance $\mathcal{I}_2$ has a solution.

Suppose now that $\mathcal{I}_2$ has a solution. Let L be the latency of C_{n+1} and I be its set of predecessors. The plan is a forest, which means that the services of I are chained. We prove as in the previous computation that

$$L \geq (K + (\frac{3}{2A(A-S)})(\frac{S}{2} - \sum_{i \in I} x'_i)^2) - 2n3^n\beta^n\frac{x_M^3}{A^3}$$

By hypothesis, we have $L \leq K$. Hence:

$$\begin{array}{ll} & (\frac{3}{2A(A-S)})(\frac{S}{2} - \sum_{i \in I} x'_i)^2) \leq 2n3^n\beta^n\frac{x_M^3}{A^3} \\ \iff & (\frac{S}{2} - \sum_{i \in I} x'_i)^2 \leq 4n3^n\beta^n\frac{x_M^3(A-S)}{3A^2} \end{array}$$

By construction of A, we have $4n3^n\beta^n\frac{x_M^3(A-S)}{3A^2} \leq 4n3^n\beta^n\frac{x_M^3}{3A} < 1$. This proves that $(\frac{S}{2} - \sum_{i \in I} x'_i)^2 = 0$. Then I is a valid solution for the instance $\mathcal{I}_1$. This concludes the proof. ■

Summary

We have identified in this section three realistic communication models and we have addressed the following problems:

– Given an execution graph, what is the complexity of computing the period and the latency?
– What is the complexity of the general period or latency minimization problem?

Sections 2.3 and 2.4 give a complete set of complexity results on all variants of the one-to-one mapping problem of filtering services. In [70], the problem of scheduling filtering applications was studied in the case of linear platforms. In the next section, we extend results of this paper to several variants of the problem.

2.5 Problems on a linear heterogeneous platform

In this section, we extend the results of [70] in another important direction: we investigate the situation where services are no longer independent but instead where they are ordered along a linear chain of precedence. In this case, both services and processors are arranged according to a fixed prescribed order. This problem is the extension of the well known chains-to-chains problem [61] to the case where nodes have a selectivity, and it has a great practical significance because linear dependence chains are ubiquitous in workflow applications (see [73, 74] and the references therein).

We first describe the different variants of this problem studied in this section. Then, we prove the complexity of all this variants.

2.5.1 Framework

This section is devoted to a precise statement of these optimization problems.

The basic network topology that we consider is a linear chain of m processors $S_1, ..., S_p$. Processor S_u can only send data to S_{u+1}, for $1 \leq u \leq p-1$. This corresponds to a hierarchical network, where S_1 is the processor acquiring the data. Processor S_p is at the top of the hierarchy, and outputs the tuples of each data set that were processed through all services.

We define below the different variants of the problem.

Service ordering

The more flexible problem is the case with no precedence constraints as in [70]: services are independent, and they can be applied on the data in any order. This problem is called *Free* ordering.

We also consider the case in which services are totally ordered along a linear dependence chain: there is a precedence constraint from C_i to C_{i+1} for $1 \leq i \leq n-1$. We note this problem as the *Ordered* instance.

Service costs

The cost of executing a service depends (i) upon the processor it is assigned to and (ii) upon the combined selectivity of its predecessors. As for (i), each service has a different cost on each processor: the execution of service C_i on processor S_u takes time $C_{i,u}$. These costs may be *arbitrary*, or in some cases they take the form $C_{i,u} = \frac{w_i}{s_u}$: they are *proportional* to an amount of work w_i required by the service, and inversely proportional to the speed s_u of the processor. In the latter case, two different services have the same execution time ratios on two different processors; proportional costs are also called *uniform* costs in the scheduling literature [15]. As for (ii), the cost of executing a service is modified by all its predecessors: if $\mathsf{pred}(C_i)$ denotes the set of all predecessors of C_i in the mapping, then its execution cost on processor S_u is $\left(\prod_{C_j \in \mathsf{pred}(C_i)} \sigma_j\right) \times w_{i,u}$. Basically, we see there are two ways to decrease the final cost of a service: (i) map it on a server that executes it fast; and (ii) map it as a successor of services with small selectivities.

The execution of service C_i on processor S_u takes a time $C_{i,u}$. In the most general instance, these costs are *Arbitrary*.

However, for uniform machines, costs $C_{i,u}$ take the form $C_{i,u} = \frac{w_i}{s_u}$, where w_i is the amount of work required by the service, and s_u is the speed of processor S_u. We refer to such costs as *Proportional* costs.

Communication costs

We consider two models of platforms, with or without communication costs. For the model with communication costs, we use the same framework as [70]. They consider a model without computation/communication overlap: a server cannot compute some data and communicate with another server at the same time, these actions are serialized.

Let $\mathcal{F}_u$ denote the set of services that are mapped on processor S_u. Let PRED_u be the set of services mapped on processors S_v before S_u:

$$\text{PRED}_u = \{C_j \mid \exists v < u, alloc(C_j) = S_v\}$$

Equivalently, $\text{PRED}_u = \bigcup_{v=1}^{u-1} \mathcal{F}_v$. Finally, let $\mathcal{G}_u$ denote the set of services that are mapped before S_u, plus those mapped onto S_u:

$$\mathcal{G}_u = \text{PRED}_u \cup \mathcal{F}_u$$

The communication cost between servers S_u and S_{u+1} is given by the value

$$C_{comm}(u) = l(u) \times \prod_{C_j \in \mathcal{G}_u} \sigma_j$$

where $l(u)$ is the inverse of the bandwidth of the link from S_u to S_{u+1}. Indeed, the output of S_u is filtered by all services mapped before S_u, and by those mapped on S_u, it is thus the set $\mathcal{G}_u$. We take

into account the cost $C_{comm}(0)$ of input for processor S_1 and the cost $C_{comm}(p)$ of output for processor S_m. The corresponding bandwidths $l(0)$ and $l(p)$ corresponds to the communication links between the platform and the external world (the user).

The model with communication costs is denoted by *Cost* and the model without by *NoCost*.

Objective function

As Sections 2.3 and 2.4, the criteria considered are the period and the latency. The context of chain of tasks however simplifies the formula used to computes this criteria.

Formally, we define the period and the latency using $\mathcal{F}_u$, PRED_u, and $\mathcal{G}_u$, which correspond to the sets of services mapped on, before, and up to S_u respectively. Note that $\text{PRED}_u \subset \mathsf{pred}(C_j) \subset \mathcal{G}_u$ for each service $C_j \in \mathcal{F}_u$: $\mathsf{pred}(C_j)$, the predecessors of C_j, are all services mapped onto preceding processors, plus those mapped on S_u before C_j. To simplify notations, suppose that services in $\mathcal{F}_u$ are placed in order $C_1 \to C_2 \to ... \to C_k$. We obtain the following computation cost $C_{comp}(u)$ for processor S_u:

$$C_{comp}(u) = \left(\prod_{C_j \in \text{PRED}_u} \sigma_j \right) \sum_{i=1}^{k} \left(\prod_{q=1}^{i-1} \sigma_q \right) \times C_{i,u}$$

For a model without communication cost, $C_{comp}(u)$ is the cycle-time of processor S_u. The period is

$$\mathcal{P} = \max_{1 \leq u \leq p} \{C_{comp}(u)\}$$

and the latency is

$$\mathcal{L} = \sum_{u=1}^{p} C_{comp}(u)$$

For a model with communication cost, we need to take into account $C_{comm}(u)$. Since we consider a model with no overlap, computations and communications are serialized and we obtain a period

$$\mathcal{P} = \max_{1 \leq u \leq p} \{C_{comm}(u-1) + C_{comp}(u) + C_{comm}(u)\}$$

and a latency

$$\mathcal{L} = C_{comm}(0) + \sum_{u=1}^{p} (C_{comp}(u) + C_{comm}(u))$$

Taxonomy of problems

The taxonomy of problems differs of that of the previous sections because of the number of parameters used. We denote each problem by $Obj - XYZ$, where:

- $Obj = \text{MINPERIOD}|\text{MINLATENCY}$ denotes the objective function.
- $X = O|F$ denotes the service ordering (*Ordered* or *Free*);
- $Y = P|A$ denotes the service costs (*Proportional* or *Arbitrary*);
- $Z = C|N$ denotes the communication costs (*Cost* or *NoCost*);

For instance, MINPERIOD-FAC is the problem of minimizing the latency with no precedence constraints between services, arbitrary service costs, and with communication costs.

In addition, * denotes any instance of the problem, thus MINLATENCY-F** denotes the problem of minimizing the latency with no precedence constraints between services, for any kind of service and communication costs.

2.5.2 Period minimization

In this section we prove the NP-completeness of problems MINPERIOD-F** (all problems with free ordering), and we present a polynomial algorithm for problems MINPERIOD-O** (all problems with fixed ordering).

Free ordering

Theorem 2.10. *All problems* MINPERIOD-*F*** *are NP-hard.*

Proof. We show that MINPERIOD-FPN is NP-hard. All other problems are more difficult instances since *Proportional* is a particular case of *Arbitrary*, and *NoCost* a particular case of *Cost*.

The proof is straightforward. 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 reduction from 2-PARTITION [34]. Let $\mathcal{I}_1$ be an instance of 2-PARTITION: given a set $X = \{x_1, ..., x_n\}$, does there exist a subset I such that $\sum_{x_i \in I} x_i = \frac{1}{2} \sum_{x_j \in X} x_j$? We construct the instance $\mathcal{I}_2$ with n services and 2 servers such that:

- $\forall 1 \leq i \leq n, \sigma_i = 1$
- $\forall 1 \leq i \leq n, w_i = x_i$
- $s_1 = s_2 = 1$
- $K = \frac{1}{2} \sum_{x_j \in X} x_j$

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. Suppose that $\mathcal{I}_1$ has a solution I. We construct $alloc$ such that: $\forall i, alloc(i) = 1 \iff x_i \in I$. Then, the period of the mapping is $P = \max\{\sum_{x_i \in I} x_i, \sum_{x_i \notin I} x_i\}$, that means $P = K$. Then, $\mathcal{I}_2$ has a solution.

Suppose now that $\mathcal{I}_2$ has a solution. Let $I = \{x_i | alloc(C_i) = S_1\}$. By hypothesis, we have $\sum_{x_i \in I} x_i \leq K$ and $\sum_{x_i \notin I} x_i = 2K - \sum_{x_i \in I} x_i \leq K$. We can conclude that $\sum_{x_i \in I} x_i = \frac{1}{2} \sum_{x_j \in X} x_j$. Then, $\mathcal{I}_1$ has a solution. This concludes the proof. ■

Fixed ordering

Theorem 2.11. *Algorithm 6 computes the optimal mapping for problem* MINPERIOD-*OAC in time* $O(m \times n^3)$.

Proof. Let $\mathcal{I}$ be an instance of MINPERIOD-OAC. We prove by induction that for any pair (i, j), the value $P(i, j)$ returned by Algorithm 6 is the optimal period on the instance $\mathcal{I}_{i,j}$ restricted to the last i services and the last j servers. Moreover, $alloc(i, j, .)$ is the corresponding allocation function.

First, we compute the values $P(0, j)$ and $P(i, 1)$ for $1 \leq j \leq m$ and $1 \leq i \leq n$. In these cases, there is only one possible mapping: for $P(0, j)$, there are no services to map; for $P(i, 1)$, all services must be mapped onto the last server. Thus the computed period is optimal.

Now we consider the placement of the remaining services. Suppose that for all $j' < j$ and for all i, $P(i, j')$ is optimal. Then we show that $P(i, j)$ also is optimal. We define, for all $0 \leq r \leq i$, $f(r)$ as the period obtained by placing the r first services on server $m - j + 1$ and the other services optimally onto the next servers. We prove that the minimum of the values $f(r)$ is the optimal value for $P(i, j)$. Let $alloc^*$ be an allocation of the last i services on the last j servers and P^* be the period of this mapping. Let $S = \{i \mid alloc^*(i) = p - j + 1\}$, and $k = |S|$. Let P' be the period on $alloc^*$ for the last $i - k$

Algorithm 6: Optimal algorithm for MINPERIOD-OAC.

Data: n services of selectivities $\sigma_1, ..., \sigma_n$,
p servers with a matrix of costs C, and
a vector of communication costs l
Result: a mapping G optimizing the latency

$P(0,1) = l(p-1) + l(p)$;
for $j = 2$ **to** p **do**
  $P(0,j) = \max\{l(p-j) + l(p-j+1), P(0,j-1)\}$;
for $i = 1$ **to** n **do**
  $P(i,1) = l(p-1) + C_{n-i+1,p} +$
    $\sigma_{n-i+1}(P(i-1,1) - l(p-1))$;
  $\forall 1 \leq k \leq i,\ alloc(i,1,n-k+1) = p$;
for $j = 2$ **to** p **do**
  for $i = 1$ **to** n **do**
    $\forall\ 0 \leq r \leq i,\ f(r) = \max\{l(p-j) +$
      $\sum_{q=1}^{r} \prod_{u=1}^{q-1} \sigma_{n-i+u} C_{n-i+q,p-j+1} +$
      $\prod_{u=1}^{r} \sigma_{n-i+u} l(p-j+1),$
      $\prod_{u=1}^{r} \sigma_{n-i+u} P(i-r, j-1)\}$;
    $k = argmin_{1 \leq r \leq i}\{f(r)\}$;
    $P(i,j) = f(k)$;
    $\forall 1 \leq q \leq k,$
      $alloc(i,j,n-i+q) = p-j+1$;
    $\forall n-i < q < n-i+k,$
      $alloc(i,j,q) = alloc(i-k, j-1, q)$

services on the last $j-1$ servers. By the hypothesis, $P' \geq P(i-k, j-1)$ and

$$\begin{aligned} P(i,j) \leq & \max\{l(j) + \textstyle\sum_{i' \in S} \prod_{q \in S, q < i'} \sigma_q \times C_{i',p-j+1} \\ & + \textstyle\prod_{q \in S} \sigma_q l(j+1), \prod_{q \in S} \sigma_q P(i-k, j-1)\} \\ \leq & \max\{l(j) + \textstyle\sum_{i' \in S} \prod_{q \in S, q < i'} \sigma_q \times C_{i',p-j+1} \\ & + \textstyle\prod_{q \in S} \sigma_q l(j+1), \prod_{q \in S} \sigma_q P'\} \\ \leq & P^* \end{aligned}$$

Since this is true for any mapping leading to a period P^*, $P(i,j)$ is the optimal period. We can conclude that $P(n,p)$ is the optimal period for instance $\mathcal{I}$. ■

Corollary 2.1. *Problems* MINPERIOD-*O*** *have polynomial complexity.*

Corollary 2.2. *Problems* MINPERIOD-*O*** *have polynomial complexity.*

Proof. The most difficult problem of MINPERIOD-O** is MINPERIOD-OAC, which is polynomial due to Theorem 2.11. ■

2.5.3 Latency minimization

In this section, we present a polynomial algorithm for problems MINLATENCY-O** (fixed ordering) and we prove the NP-completeness of problems MINLATENCY-FA*. Recall that problems MINLA-

TENCY-FP* are showed to be polynomial in [70]. With arbitrary costs instead of proportional costs, the problem becomes NP-hard, even in the absence of communications.

Fixed ordering

We derive an optimal algorithm for problems MINLATENCY-OAN and MINLATENCY-OAC. The algorithm for MINLATENCY-OAN (without communications) is presented only because it is simpler to understand than the algorithm for MINLATENCY-OAC (with communications). The complexity is the same for both cases.

Algorithm 7: Optimal algorithm for MINLATENCY-OAN.

Data: n services of selectivities $\sigma_1, ..., \sigma_n \leq 1$ and p servers with a matrix of costs C
Result: a mapping G optimizing the latency

for $j = 1$ **to** p **do**
  $L(0, j) = 0$;
for $i = 1$ **to** n **do**
  $L(i, 1) = C_{n-i+1,p} + \sigma_{n-i+1} L(i-1, 1)$;
  $\forall 1 \leq k \leq i,\ alloc(i, 1, n-k+1) = p$;
for $j = 2$ **to** p **do**
  for $i = 1$ **to** n **do**
    $\forall\ 0 \leq l \leq i,\ f(l) =$
      $\sum_{i'=1}^{l} \left(\prod_{q=1}^{i'-1} \sigma_{n-i+q} \right) C_{n-i+i',p-j+1} +$
      $\left(\prod_{q=1}^{l} \sigma_{n-i+q} \right) L(i-l, j-1)$;
    $k = argmin_{0 \leq l \leq i}\{f(l)\}$;
    $L(i, j) = f(k)$;
    $\forall 1 \leq q \leq k,$
      $alloc(i, j, n-i+q) = p-j+1$;
    $\forall k < q \leq i, \quad alloc(i, j, n-i+q) = alloc(i-k, j-1, n-i+q)$;

Theorem 2.12. *Algorithm 7 computes the optimal mapping for problem* MINLATENCY-*OAN in time* $O(n^3 m)$.

Proof. Let $\mathcal{I}$ be an instance of MINLATENCY-OAN. We prove by induction that for any pair (i, j), the value $L(i, j)$ returned by Algorithm 7 is the optimal latency on the instance $\mathcal{I}_{i,j}$ restricted to the last i services and the last j servers. Moreover, $alloc(i, j, .)$ is the corresponding allocation function.

First, we compute the values $P(0, j)$ and $P(i, 1)$ for $1 \leq j \leq m$ and $1 \leq i \leq n$. In these cases, there is only one possible mapping: either there are no services to map, or all services must be mapped onto the last server. Thus the computed latency is optimal.

Suppose that for all $j' < j$ and for all $1 \leq i \leq n$, $L(i, j')$ is optimal. Then we prove that for all i, $L(i, j)$ also is the optimal latency. Let $alloc^*$ be an allocation of the last i services on the last j servers and L^* be the latency of this mapping. Let $S = \{i \mid alloc^*(i) = p - j + 1\}$, and $k = |S|$. Let L' be the latency on $alloc^*$ for the last $i-k$ services on the last $j-1$ servers. By hypothesis, $L' \geq L(i-k, j-1)$, and
$L(i, j) \leq f(k)$

$\leq \sum_{i' \in S} \prod_{q \in S, q<i'} \sigma_q \times C_{i',p-j+1}$
$\quad +(\prod_{q \in S} \sigma_q) L(i-k, j-1)$
$\leq \sum_{i' \in S} \prod_{q \in S, q<i'} \sigma_q \times C_{i',p-j+1} + (\prod_{q \in S} \sigma_q) L'$
$\leq L^*$

Since this is true for any mapping leading to a latency L^*, $L(i,j)$ is the optimal latency. We can conclude that $L(n,p)$ is the optimal latency for instance $\mathcal{I}$. ■

Algorithm 8: Optimal algorithm for MINLATENCY-OAC.

Data: n services of selectivities $\sigma_1, ..., \sigma_n$, p servers with a matrix of costs C and a vector of communication cost l
Result: a mapping G optimizing the latency

for $j = 1$ **to** p **do**
  $L(0,j) = \sum_{j'=p-j+1}^{p} l(j')$;
for $i = 1$ **to** n **do**
  $L(i,1) = l(p-1) + C_{n-i+1,p} + \sigma_{n-j+1}(L(i-1,1) - l(p-1))$;
  $\forall 1 \leq k \leq i,\ alloc(i,1,n-k+1) = p$;
for $j = 2$ **to** p **do**
  for $i = 1$ **to** n **do**
    $\forall\ 0 \leq l \leq i, \quad f(l) =$
    $l(p-j+1) + \sum_{i'=1}^{l} \left(\prod_{q=1}^{i'-1} \sigma_{n-i+q} \right) C_{n-i+i',p-j+1} + \left(\prod_{q=1}^{l} \sigma_{n-i+q} \right) L(i-l, j-1)$;
    $k = argmin_{0 \leq l \leq i}\{f(l)\}$;
    $L(i,j) = f(k)$;
    $\forall 1 \leq q \leq k,$
      $alloc(i,j,n-i+q) = p-j+1$;
    $\forall k < q \leq i, \quad alloc(i,j,n-i+q) = alloc(i-k, j-1, n-i+q)$;

Theorem 2.13. *Algorithm 8 compute the optimal mapping for problem* MINLATENCY-*OAC in time* $O(n^3 p)$.

Proof. The proof is similar to that for Theorem 2.12. We merely add communication costs in the equations. ■

Free ordering

This section is devoted to assessing the most difficult complexity result of this chapter: the NP-completeness of latency minimization with arbitrary costs, even without taking communications into account. This important result closes the open question raised in [70].

Theorem 2.14. *Problem* MINLATENCY-*FAN is NP-complete.*

Proof. We consider the associated decision problem: given a latency K, is there a mapping of latency less than K? The problem is obviously in NP: given a latency and a mapping, it is easy to check in polynomial time whether it is valid or not.

The NP-completeness is obtained by reduction from 2-PARTITION [34], as in Theorem 2.10, but the reduction is quite involved. Let $\mathcal{I}_1$ be an instance from 2-PARTITION: given a set $X = \{x_1, ..., x_n\}$,

does it exist a subset I such that $\sum_{x_i \in I} x_i = \frac{1}{2} \sum_{x_j \in X} x_j$? Let $x_M = \max_{x_i \in X}\{x_i\}$, $S = \sum_{x_j \in X} x_j$, $\beta = \frac{A-S}{2A+S}$ and $A > \frac{4}{3} n 3^n \times x_M^3$. We construct the instance $\mathcal{I}_2$ with $n+1$ services and 3 servers such that:

- $\forall i \leq n, C_{i,1} = \frac{x_i}{A}$
- $\forall i \leq n, C_{i,2} = 3\left(\frac{3A}{A-x_M}\right)^n$
- $\forall i \leq n, C_{i,3} = 0$
- $\forall i \leq n, \sigma_i = 1 - \frac{x_i}{A} + \beta \frac{x_i^2}{A^2}$
- $C_{n+1,1} = C_{n+1,3} = 3\left(\frac{3A}{A-x_M}\right)^n$
- $C_{n+1,2} = \frac{2A+S}{2A-2S}$
- $\sigma_{n+1} = 1$
- $K = C_{n+1,2} - \frac{3S^2}{8A(A-S)} + \frac{n 3^n \beta^n x_M^3}{A^3}$

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$: the greatest value in $\mathcal{I}_2$ is A and $\log(A)$ is linear in n.

Suppose that $\mathcal{I}_1$ has a solution I. We place the services C_i with $i \in I$ in any order as a linear chain on server S_1. Then, C_{n+1} is placed on S_2, and finally the remaining services are placed on S_3. The cost of the services on S_3 is null; that means that the latency L of the system is the latency of C_{n+1}. Let $k = |I|$, and for $1 \leq i \leq k$, let c'_i be the cost of the i-th service of I on the chain on server S_1, and let σ'_i be its selectivity.

$$
\begin{aligned}
L =& \textstyle\sum_{i \leq k} \prod_{j<i} \sigma'_j c'_i + \prod_{j \leq k} \sigma'_j C_{n+1,2} \\
\leq& \textstyle\sum_{i \leq k} \frac{x'_i}{A}(1 - \sum_{j<i} \frac{x'_j}{A} + 3^n \beta^n (\frac{x_M}{A})^2) \\
& + C_{n+1,2}(1 - \textstyle\sum_{i \leq k} \frac{x'_i}{A} + \beta \sum_{i \leq k} (\frac{x'_i}{A})^2 + \sum_{i \leq k} (\frac{x'_i}{A})^2 \\
& + 2 \textstyle\sum_{i<j \leq k} \frac{x'_i x'_j}{A^2} + 3^n \beta^n \frac{x_M^3}{A^3}) \\
\leq& C_{n+1,2} + \textstyle\sum_{i \leq k} \frac{x'_i}{A}(1 - C_{n+1,2}) \\
& + \textstyle\sum_{i \leq k} (\frac{x'_i}{A})^2 C_{n+1,2}(\beta + 1) \\
& + \textstyle\sum_{i<j \leq k} \frac{x'_i x'_j}{A^2}(2C_{n+1,2} - 1) + n 3^n \beta^n \frac{x_M^3}{A^3} \\
\leq& C_{n+1,2} + \textstyle\sum_{i \leq k} x'_i (\frac{-3S}{2A(A-S)}) + \sum_{i \leq k} x_i'^2 (\frac{3}{2A(A-S)}) \\
& + \textstyle\sum_{i<j \leq k} x'_i x'_j (\frac{1}{A(A-S)}) + n 3^n \beta^n \frac{x_M^3}{A^3} \\
\leq& C_{n+1,2} + (\frac{3}{2A(A-S)})(-S \textstyle\sum_{i \leq k} x'_i \\
& + \textstyle\sum_{i \leq k} x_i'^2 + 2 \sum_{i<j \leq k} x'_i x'_j) \\
& + n 3^n \beta^n \frac{x_M^3}{A^3} \\
\leq& C_{n+1,2} + (\frac{3}{2A(A-S)})(\frac{S}{2} - \textstyle\sum_{i \leq k} x'_i)^2 - (\frac{3}{2A(A-S)}) \frac{S^2}{4} \\
& + n 3^n \beta^n \frac{x_M^3}{A^3} \\
\leq& K
\end{aligned}
$$

Then, the instance $\mathcal{I}_2$ has a solution.

Suppose now that $\mathcal{I}_2$ has a solution. By construction of $C_{n+1,1}$ and $C_{n+1,3}$, we can see that the service C_{n+1} has to be mapped onto S_2 in the solution of $\mathcal{I}_2$. Similary, there can be no service C_i ($i \leq n$) on S_2. Let L be the latency of C_{n+1} and I be its set of predecessor. Suppose that there is a service C_i with $i \in I$ on S_2, we have the latency L_i of C_i such that

$$
\begin{aligned}
L_i &\geq 3\left(\tfrac{A}{A-x_M}\right)^n \times \textstyle\prod_{i \leq n} \sigma_i \\
&> 3 \\
&> K
\end{aligned}
$$

This proves that all the services of I are mapped on S_1. We prove as in the previous computation that

$$L \geq K + \left(\frac{3}{2A(A-S)}\right)\left(\frac{S}{2} - \sum_{i\in I} x'_i\right)^2 - 2n3^n\beta^n\frac{x_M^3}{A^3}$$

By construction of A, we have

$$4n3^n\beta^n\frac{x_M^3(A-S)}{3A^2} \leq 4n3^n\beta^n\frac{x_M^3}{3A} < 1.$$

This proves that $(\frac{S}{2} - \sum_{i\in I} x'_i)^2 = 0$. Then I is a valid solution for the instance $\mathcal{I}_1$. This concludes the proof. ■

Corollary 2.3. *Problem* MINLATENCY-*FAC is NP-complete.*

Summary

In this section, we have assessed the complexity of all considered variants of the problem of mapping filtering services on linear heterogeneous platforms. Table 2.2 summarizes the complexity of all problem instances.

2.6 Conclusion

In this chapter, we have explored the problem of mapping filtering streaming applications on large-scale homogeneous and heterogeneous platforms. As in previous literature, we have essentially restricted the study to one-to-one mappings.

In a simplified model without communication cost, we have exhibited polynomial time algorithms for latency and period optimization problems on homogeneous platforms, and we have proved the NP-hardness of these problems on heterogeneous platforms. Then we have identified three natural and realistic communication models, with and without communication/computation overlap, and with one-port or bounded multi-port communications. We have addressed the following problems:

- Evaluation: given an execution graph, what is the complexity of computing the period or the latency?
- Optimization: what is the complexity of the general period or latency minimization problem?

Table 2.1 summarizes the complexity results for one-to-one mappings. We have been able to provide the complexity of all the optimization problems, thereby providing solid theoretical foundations for the study of filtering streaming applications. The evaluation of the optimal period for a given graph is polynomial with the OVERLAP model, and NP-complete for the two other models, and the evaluation of the optimal latency for a given graph is NP-complete with the three communication cost models. These results apply to regular workflow applications, which broadens the scope and significance of our results to quite a large applicative framework.

The results on linear platforms are summarized in Table 2.2. The ordered variant (O*), that corresponds to an interval mapping of a chain of tasks on a linear platform, is polynomial for both period and latency minimization. With free order of task and general mapping, the period minimization is NP-hard, when the latency minimization is polynomial when cost of tasks are proportional to speed of processors, and NP-complete with arbitrary costs.

Altogether, this chapter provides a comprehensive overview of the additional difficulties induced by heterogeneity and communication costs. In the future, we plan to explore models that allow preemption. This would require to carefully assess the cost of interruptions. Another important extension of

	Mapping already given		Optimization problem	
	Period	Latency	Period	Latency
Hom. without comm.	Polynomial	Polynomial	Polynomial	Polynomial
Het. without comm.	Polynomial	Polynomial	NP-hard Inapproximable	NP-hard Inapproximable
Hom. with comm.	OVERLAP: Polynomial Other models: NP-hard	NP-hard	NP-hard	NP-hard

Table 2.1: Complexity results for one-to-one mappings.

model	MINPERIOD	MINLATENCY
Linear applications	Polynomial	Polynomial
General applications with proportional costs	NP-hard	Polynomial
General applications with arbitrary costs	NP-hard	NP-hard

Table 2.2: Complexity results for linear platforms.

this work would be to tackle bi-criteria problems with communication costs. In addition, we plan to search for approximation algorithms and lower bounds, or at least efficient heuristics, for all NP-hard problem instances. Allowing some services to be replicated would allow to decrease the period of the mappings, while data-parallelizing some other services would allow to decrease both period and latency. To the best of our knowledge, such extensions, which are well-known and widely used in the context of classical pipelined workflows, have never been addressed for filtering services. This is an interesting but algorithmically challenging direction to explore.

Chapter 3

Reliability and performance optimization of pipelined real-time systems

3.1 Introduction

Filtering applications studied in the previous chapter are a generalization of the workflow model. The selectivity value of a task corresponds to its probability of success. We consider in this chapter the problem of linear workflow reliability.

Pipelined *real-time systems* are commonly found in assembly lines and are subject to strict *dependability* and *real-time constraints*. They consist of a chain of tasks executing on a distributed platform. Each task is a block of code with a known amount of work to be processed. The role of the first task of the chain is to acquire some data set from the environment (thanks to sensor drivers), to process it, and finally to transmit its result to the second task. Each subsequent task receives its input data from its immediately preceding task, processes it, and transmits its result to its immediately successor task, except the last task that transmits it to the environment (thanks to actuator drivers).

Tasks are assigned to processors of the platform using an *interval mapping*, which groups consecutive tasks of the linear chain and assigns them to the same processor. Interval mappings are more general than one-to-one mappings, which establish a unique correspondence between tasks and processors; they are very useful for reducing communication overheads, not to mention the many situations where there are more tasks than processors and where interval mappings are mandatory. The key performance-oriented metrics to determine the best interval mapping are the *period* and the *latency*. The period is the time interval between the beginning of the execution of two consecutive data sets. Equivalently, the inverse of the period is the *throughput*, which measures the aggregate rate of processing of data. 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 for processing the data set entirely. Minimizing the latency is *antagonistic* to minimizing the period, and trade-offs should be found between these two criteria.

Besides real-time constraints, expressed as an upper bound on the period and/or the latency, pipelined real-time systems must also satisfy crucial *dependability constraints*, which are expressed as a lower bound on the *reliability* of the mapping. Increasing the reliability is achieved by replicating the intervals on several processors. Augmenting the replication level (defined as the average number of times each interval is replicated) is good for the reliability, but bad for the period and latency, because less processors will be available for executing the intervals of tasks. We thus have three antagonistic criteria: reliability, period, and latency.

We evaluate the reliability of a single task mapped onto a processor according to the classical model of Shatz and Wang [67], where each hardware component (processor or communication link) is *fail-*

silent and is characterized by a *constant failure rate per time unit* λ: the reliability of a task of duration d is therefore $e^{-\lambda d}$. For an interval of several tasks mapped onto a single processor, we just have to sum up the task durations, hence obtaining $e^{-\lambda D}$, where D is the sum of the interval's task durations. For a mapping with replication, we compute the reliability by building the *Reliability Block Diagram* (RBD) [56, 7] corresponding to this mapping. Here we face the delicate issue that computing the reliability is exponential in the size of the mapping (or equivalently the size of the RBD). To solve this issue, we insert *routing operations* in the mapping to guarantee that the RBD is by construction serial-parallel, therefore allowing us to compute its reliability in linear time.

The models are detailed in Section 3.2 and we discuss related work in Section 3.3.

Our contribution is multifold. In Section 3.4, we show how to compute the different objectives (reliability, expected and worst-case latency, and expected and worst-case period) for a given multiprocessor mapping.

Then, we derive complexity results for homogeneous platforms in Section 3.5. We prove that:

1. computing a mono-criterion mapping that optimizes the reliability is *polynomial* (Section 3.5.1);
2. optimizing both the reliability and the period remains *polynomial* (Section 3.5.2);
3. the problem of optimizing both the reliability and the latency is *NP-complete* (Section 3.5.3);
4. the problem of assigning processors for a given partition of the chain of task in intervals is *polynomial* (Section 3.5.5).

Moreover, for homogeneous platforms, a linear program is provided to solve the problem of optimization of reliability for given bounds on period and latency in Section 3.5.4.

For heterogeneous platforms, we prove that the mono-criterion problem of optimizing the reliability is *NP-complete*, and hence all the multi-criteria mapping problems that include the reliability in their criteria are also *NP-complete* (Section 3.6).

We provide heuristics in Section 3.7 for the more general problem of optimizing the reliability under constraints on period and latency on a heterogeneous platform, and we conduct experiments on homogeneous and heterogeneous platforms to assess their performance (Section 3.8).

Finally, we state some concluding remarks and future research directions in Section 3.9.

3.2 Framework

In this section, we detail the application model, the platform model, the failure model, and the replication model. We end with the formal definition of the mono- and multi-criteria multiprocessor mapping problems.

3.2.1 Application model

An application is a *chain* of n tasks $\mathcal{C} = (\tau_i)_{1 \leq i \leq n}$. Each task τ_i is a block of code that (1) receives its input from its predecessor τ_{i-1}, (2) computes a known amount of work, (3) and produces an output data set of a known size. Therefore, each task τ_i is represented by the pair (w_i, o_i), where w_i is the amount of work and o_i is the output data size. By convention, $o_n = 0$ because τ_n emits its result directly to the environment through actuator drivers. Specifying the size of the input data set required by a task is not necessary since, by definition of a chain, it is equal to the size of the output data set of its immediately preceding task. Figure 3.1 shows an example of a chain composed of n tasks.

Executing τ_i on a processor of speed s takes w_i/s units of time. Transmitting the result of τ_i on a link of bandwidth b takes o_i/b units of time. Knowing the values w_i and o_i is not a critical assumption since

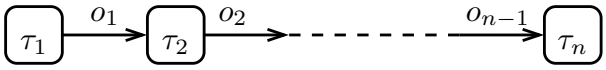

Figure 3.1: Example of a chain of n tasks.

worst-case execution time (WCET) analysis has been applied with success to real-life processors actually used in embedded systems. In particular, it has been applied to the most critical existing embedded system, namely the Airbus A380 avionics software running on the Motorola MPC755 processor [30, 69].

3.2.2 Platform model

The target platform consists of p processors connected by point-to-point communication links. We note $\mathcal{P}$ the set of processors: $\mathcal{P} = (P_u)_{1 \leq u \leq p}$. We assume that communication links are *homogeneous*: this means that all links have the same bandwidth b. On the contrary, each processor P_u may have a different speed s_u. Such platforms correspond to networks of workstations with plain TCP/IP interconnects or other LANs.

In order to derive a realistic communication model, we assume that the number of outgoing point-to-point connections of each processor is limited to $\mathcal{K}$. A given processor is thus capable of simultaneously sending messages to (and receiving messages from) $\mathcal{K}$ other processors. Indeed, there is no physical device capable of sending, say, 100 messages to 100 distinct processors, at the same speed as if it was a single message. The output bandwidth of the sender's network card would be a limiting factor. Our assumption of bounded multi-port communications [48] is reasonable for a large range of platforms, from large-scale clusters to multi-core System-on-Chips (SoCs).

In addition, we assume that communications are *overlapped* with computations, that is, a processor can compute the current instance of task τ_i and, in parallel, send to another processor the result of the previous instance of τ_i. This model is consistent with current processor architectures where a SoC can include a main processor and several communication co-processors.

3.2.3 Interval mapping

The chain of tasks is executed repeatedly in a *pipelined manner* to achieve a better throughput. As a consequence, mapping the chain on the platform involves dividing the chain into m intervals of consecutive tasks, and assigning each processor to a unique interval. This technique is known as *interval mapping*. Figure 3.2 shows an example of a division of a chain of tasks into m intervals.

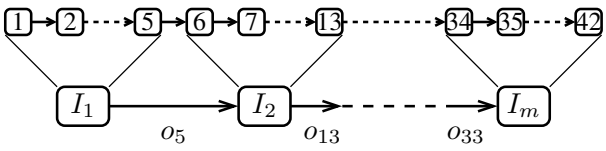

Figure 3.2: A chain of tasks divided into m intervals.

In a mapping without replication, each interval is assigned to a single processor. If the number of processors is greater than the number of tasks, then each interval can be of size one (that is, one task per interval), but this is rarely the case for real-life systems. Furthermore, having many small intervals

is likely to decrease the period and the failure probability, but it will also increase the communication costs, and hence the latency: thus a trade-off is to be found.

In a mapping with replication, each interval is assigned to several processors. Replication is crucial to increase the reliability of the system [35].

For each $1 \leq j \leq m$, the interval I_j is the set of consecutive tasks between indices f_j and l_j. Moreover, $f_1 = 1$, $\forall 2 \leq j \leq m, f_j = l_{j-1} + 1$, and $l_m = n$. The amount of work processed by I_j is therefore $W_j = \sum_{\tau_i \in I_j} w_i = \sum_{i=f_j}^{l_j} w_i$. The size of the output data set produced by interval I_j is that of its last task, that is, o_{l_j}.

3.2.4 Failure model

Both processors and communication links can fail, and they are *fail-silent*. Classically, we adopt the failure model of Shatz and Wang [67]: failures are *transient* and the maximal duration of a failure is such that it affects only the current operation executing onto the faulty processor, and not the subsequent operations (same for communication links); this is the "hot" failure model. Besides, the occurrence of failures on a processor (same for a communication link) follows a Poisson law with a constant parameter λ, called its *failure rate per time unit*. Modern fail-silent hardware components can have a failure rate around 10^{-6} per hour.

Since communication links are homogeneous, we note λ_ℓ their identical failure rate per time unit. Concerning the processors, we note λ_u the failure rate per time unit of the processor P_u, for each P_u in $\mathcal{P}$.

Moreover, failure occurrences are *statistically independent events*. Note that transient failures are the most common failures in modern processors, all the more when processor voltage is lowered to reduce the energy consumption, because, in that case, even very low energy particles are likely to create a critical charge leading to a transient failure [90].

The *reliability* of a system measures its continuity of service. It is defined as the probability that it functions correctly during a given time interval [4]. According to our model, the reliability of the processor P (resp. the communication link L) during the duration d is $r = e^{-\lambda d}$, where λ is the failure rate per time unit of P or L. Conversely, the *probability of failure* of the processor P (resp. the communication link L) during the duration d is $f = 1 - r = 1 - e^{-\lambda d}$. Hence, the reliability of the task τ_i on processor P_u is:

$$r_{u,i} = e^{-\lambda_u \, w_i \, / \, s_u} \, . \tag{3.1}$$

Accordingly, the reliability of the interval I mapped on the processor P_u is:

$$r_{u,I} = e^{-\lambda_u \, W_j \, / \, s_u} = \prod_{\tau_i \in I} r_{u,i} \, . \tag{3.2}$$

Equations (3.1) and (3.2) show that platform heterogeneity may come from two factors: (i) processors having different speeds, and (ii) processors having different failure rates. We say that the platform is *homogeneous* if all the processors have the same speed s and the same failure rate λ (hence the reliability and the execution time of an interval no longer depends on the processor it is assigned to, and we use in this case the notation r_i instead of $r_{u,i}$ in Equation (3.1)); otherwise, we say that the platform is *heterogeneous*.

Finally, we let $r_{comm,i} = e^{-\lambda_\ell \, o_i \, / \, b}$ denote the reliability of the i-th communication.

3.2.5 Replication model

We use *spatial redundancy* to increase the reliability of a system: in other words, we replicate the intervals on several processors. Figure 3.3 shows an example of mapping by interval with spatial redundancy: the interval I_1 is mapped on the processors $\{P_1, P_2, P_3\}$, the interval I_2 is mapped on the processors $\{P_4, P_5\}$, and so on until the interval I_m, which is mapped on the processors $\{P_{p-1}, P_p\}$. Concerning the communications, the data-dependency o_{l_1} is mapped on the point-to-point links $\{L_{14}, L_{15}, L_{24}, L_{25}, L_{34}, L_{35}\}$, and so on.

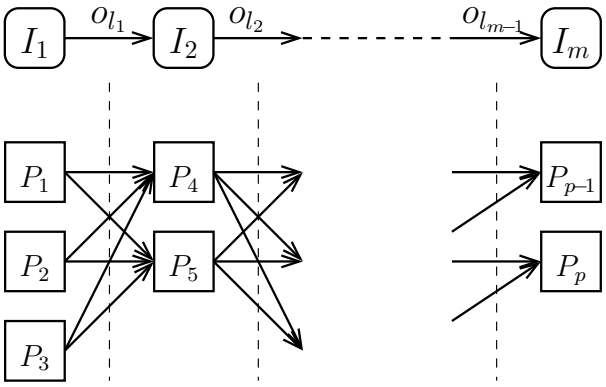

Figure 3.3: An example of interval mapping.

To increase the reliability, *each* processor of a given interval communicates with *each* processor of the next interval. Specifically, for any $1 \leq j \leq m-1$, all the processors executing interval I_j send their result to all the processors executing the next interval I_{j+1}. Because of the bounded number $\mathcal{K}$ of possible communications (see Section 3.2.2), the maximum number of replicas per interval is also limited to $\mathcal{K}$.

3.2.6 Multiprocessor mapping problem

We study several variants of the multiprocessor interval mapping problem. The inputs of the problem are a chain of n tasks $\mathcal{C} = (\tau_i)_{1 \leq i \leq n}$, a hardware platform of p processors $\mathcal{P} = (P_u)_{1 \leq u \leq p}$, and a bound $\mathcal{K}$ on the maximal number of replications for each interval of tasks. The output is an interval mapping of $\mathcal{C}$ onto $\mathcal{P}$, that is, a distribution of $\mathcal{C}$ into m intervals and an assignment of each interval to at most $\mathcal{K}$ processors of $\mathcal{P}$, such that each processor executes only one interval. Each variant of the mapping problem optimizes a different set of criteria among the following:

- the reliability,
- the expected input-output latency,
- the worst-case input-output latency,
- the expected period,
- the worst-case period.

3.3 Related work

Several papers have dealt with workflow applications the dependence graph of which is a linear chain. The pioneering papers [73, 74] investigate bi-criteria (period, latency) optimization of such work-

flows on homogeneous platforms. An extension of these results to heterogeneous platforms is provided in [10, 11].

All the previous papers deal with fully reliable platforms. In our previous work [9], we studied the (reliability, latency) mapping problem with fail-silent processors. The model in [9] is quite different, and much more crude, than the one of this chapter: each processor has an absolute probability of failing, independent of task durations, and the faults are unrecoverable. To the best of our knowledge, we are not aware of other published work on optimizing linear workflows for reliability. However, many papers have dealt with a directed acyclic graph (DAG) instead of a pipelined workflow, be it a fully general DAG [25], a linear chain [66], or even independent tasks [50, 66]. The closest to our present work is [66]: it contains a short section on linear chains, with a mono-criterion dynamic programming algorithm for optimizing the reliability, which is similar to our Algorithm 9 (see Section 3.5.1).

Finally, the specific problem of bi-criteria (length, reliability) multiprocessor scheduling has also been addressed in [24, 3, 40, 63, 36, 37] for general DAGs of operations, but except [3, 36, 37], these papers do not replicate the operations and have thus a very limited impact on the reliability. Moreover, none of them consider chains of tasks and interval mappings, and therefore they attempt to minimize the length of the mapping without distinguishing between the period and the latency (the latter one being equivalent to the schedule length).

3.4 Evaluation of a given mapping

In this section, we detail the computation of the different objectives (reliability, expected and worst-case latency, and expected and worst-case period) for a given mapping. We compute the reliability of a mapping by building its *reliability block diagram* (RBD) [56, 7]. Formally, a RBD is an *acyclic oriented graph* (N, E), where each node of N is a *block* representing an element of the system, and each arc of E is a *causality link* between two blocks. Two particular connection points are its *source* S and its *destination* D. An RBD is *operational* if and only if there exists at least one operational path from S to D. A path is operational if and only if all the blocks in this path are operational. The probability that a block be operational is its reliability. By construction, the probability that a RBD is operational is equal to the reliability of the system that it represents.

In our case, the system is the multiprocessor interval mapping, possibly partial, of the application on the platform. A mapping is *partial* if not all intervals have been mapped yet, but of course those intervals that are mapped are such that all their predecessors are also mapped. Each block of the RBD represents an interval I_j placed on a processor or a data-dependency o_{l_j} between the two intervals I_j and I_{j+1} placed on a communication link. The reliability of a block is therefore computed according to Equation (3.2).

Computing the reliability in this way assumes that the occurrences of the failures are statistically independent events (see Section 3.2.4). Without this hypothesis, the fact that some blocks belong to several paths from S to D makes the computation of the reliability very complex. Concerning hardware faults, this hypothesis is reasonable, but this would not be the case for software faults [52].

The main drawback of this approach is that the computation of the reliability is, in general, exponential in the size of the RBD. When the schedule is without replication, the RBD is *serial* (i.e., there is a single path from S to D) so the computation of the reliability is linear in the size of the RBD. But when the schedule is with replications, the RBD has no particular form, so the computation of the reliability is exponential in the size of the RBD. The reason is that processors are heterogeneous: the completion dates of a given interval on its assigned processors are different, so the reception dates by the processors of the next interval are different. This is true even when the application is a chain of intervals rather than

a general graph. See Figure 3.4 for an illustration, where the RBD corresponding to the mapping has no specific form.

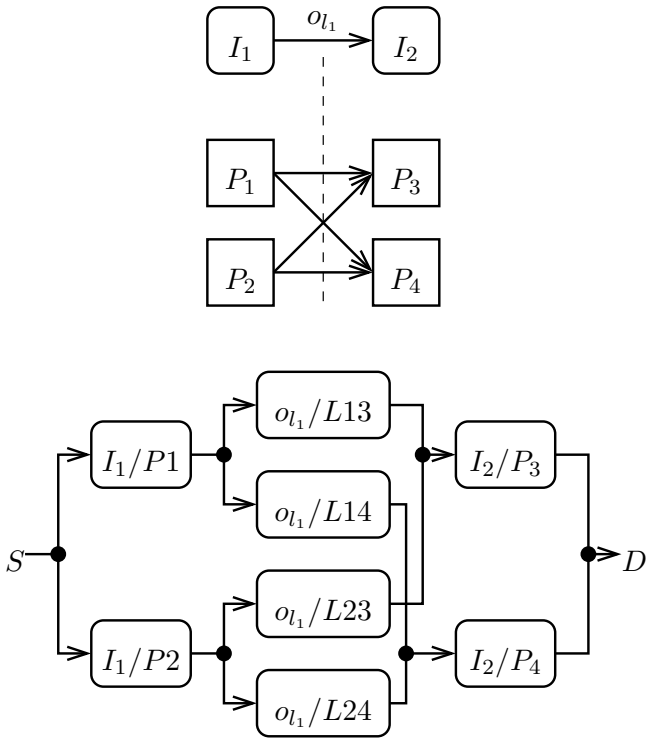

Figure 3.4: A mapping of two intervals (I_1 and I_2) on four processors (P_1 to P_4) and its RBD which has no particular form.

One solution for computing the reliability of the mapping of Figure 3.4 involves enumerating all the *minimal cut sets* of its RBD [51]. A *cut set* in a RBD is a set of blocks C such that there is no path from S to D if all the blocks of C are removed from the RBD. A cut C is *minimal* if, whatever the block that is removed from it, the resulting set is not a cut anymore. It follows that the reliability of a minimal cut set is the reliability of all its blocks put in parallel. The reliability of the mapping can then be approximated by the reliability of the alternative RBD composed of all the minimal cut sets put in sequence. Because this RBD is *serial-parallel*, this computation is linear in the number of minimal cut sets. The problem is that, in general, the number of minimal cuts is exponential in the size of the RBD [51].

For this reason, we follow the approach of [36] and we insert *routing operations* between the intervals to make sure that the RBD representing a mapping is always *serial-parallel*, therefore making tractable the computation of the reliability. This is illustrated in Figure 3.5, where a routing operation R has been mapped on processor P_5 and the RBD corresponding to the mapping is serial-parallel; as a consequence, the reliability of this mapping can be computed in a linear time w.r.t. the number of intervals.

Routing operations can be mapped on *any* processor. For instance, in the RBD of Figure 3.5, R could have been mapped on P_1 instead of P_5, therefore avoiding the need for the communication (o_{l_1}/L_{15}). Also, routing operations are assumed to be executed in 0 time units [36]. As a consequence, for any processor P_u, the reliability of the block (R/P_u) is 1.

As we have advocated, inserting routing operations yields the huge advantage of making the relia-

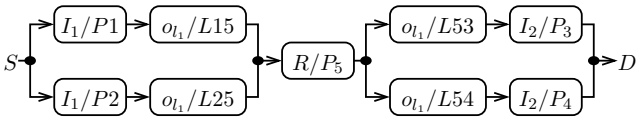

Figure 3.5: The serial-parallel RBD obtained from the same mapping as in Figure 3.4 but with an additional routing operation R.

bility computation linear in time. This comes at a cost in the execution time of the system because of the increased number of communications. For instance, in Figure 3.5, o_{l_1} is transmitted twice before reaching I_2. However, it has been shown in [36] that the overhead incurred by the routing operations is reasonable (only $+3.88\,\%$ on average).

For an interval I of weight W mapped on the subset of processors $\mathcal{P}_I$, let ec be its expected time of computation, and let wc be its worst-case execution time (by the slowest processor of $\mathcal{P}_I$). Assume that the processors in $\mathcal{P}_I$ are ordered according to their speed, from the fastest P_1 to the slowest P_t: that is, $\forall 1 \leq u < t$, we have $s_u \geq s_{u+1}$. Then, the expected and worst-case execution times of I on $\mathcal{P}_I$ are:

$$ec(I, \mathcal{P}_I) = W \times \frac{\sum_{u=1}^{t} \left(\frac{1}{s_u} r_{u,I} \prod_{v=1}^{u-1} (1 - r_{v,I}) \right)}{1 - \prod_{u=1}^{t} (1 - r_{u,I})} \; ; \tag{3.3}$$

$$wc(I, \mathcal{P}_I) = \frac{W}{s_t} \; . \tag{3.4}$$

Equation (3.3) sums up, for each P_u, the case where the first $u-1$ fastest processors fail, and the u-th one is successful. Then, for a mapping $(I_1, \mathcal{P}_1), \ldots, (I_m, \mathcal{P}_m)$, the expected latency EL and the expected period EP are:

$$EL = \sum_{i=1}^{m} ec(I_i, \mathcal{P}_i) + \frac{o_i}{b} \; ; \tag{3.5}$$

$$EP = \max \left\{ \max_{1 \leq i \leq m} \left\{ \frac{o_i}{b} \right\}, \max_{1 \leq i \leq m} ec\left(I_i, \mathcal{P}_i\right) \right\} \; . \tag{3.6}$$

The worst-case latency WL and the worst-case period WP are defined similarly, but with the worst-case cost of intervals (Equation (3.4)) instead of the expected cost (Equation (3.3)):

$$WL = \sum_{i=1}^{m} wc(I_i, \mathcal{P}_i) + \frac{o_i}{b} \; ; \tag{3.7}$$

$$WP = \max \left\{ \max_{1 \leq i \leq m} \left\{ \frac{o_i}{b} \right\}, \max_{1 \leq i \leq m} wc\left(I_i, \mathcal{P}_i\right) \right\} \; . \tag{3.8}$$

Finally, thanks to the routing operations, the reliability of the mapping $(I_1, \mathcal{P}_1), \ldots, (I_m, \mathcal{P}_m)$ is:

$$r = \prod_{i=1}^{t} \left(1 - \prod_{P_u \in \mathcal{P}_i} (1 - r_{comm,i-1} \times r_{u,I_i} \times r_{comm,i}) \right) \; . \tag{3.9}$$

Equation (3.9) above is computed according to the generic form of the RBD of Figure 3.5. To account for the fact that the first interval I_1 has no incoming communication, we just set $o_0 = 0$, hence $r_{comm,0} = 1$. The same occurs for the outgoing communication of the last interval I_m. Finally, routing operations do not appear in Equation (3.9) since their reliability is always equal to 1.

3.5 Complexity results for homogeneous platforms

In this section, we provide optimal polynomial algorithms for the mono-criterion reliability optimization problem, and then for the bi-criteria (reliability, period) optimization problem. Then, we prove the NP-completeness of the bi-criteria (reliability, latency) optimization problem. We provide an integer linear program to solve the tri-criteria problem and a polynomial time algorithm to optimally allocate processors for a given partition of the chain of tasks in intervals. Note that on homogeneous platforms, the expected latency and worst-case latency are the same. This also holds true for the expected period and worst-case period.

3.5.1 Reliability optimization

We present a mono-criterion polynomial-time algorithm that maximizes the reliability of a given chain of tasks on a given homogeneous platform. Algorithm 9 is a dynamic programming algorithm. It is a simplified version of Algorithm 10 for bi-criteria (reliability, period) optimization, which we present in the next section.

Algorithm 9: Optimal algorithm for reliability optimization on fully homogeneous platforms.

Data: a number p of fully homogeneous processors of failure rate λ, a list A of n tasks of sizes w_i, and a maximal number $\mathcal{K}$ of replications
Result: a reliability r

for $k = 1$ **to** $\min\{\mathcal{K}, p\}$ **do**
  $$F(1,k) = 1 - \left(1 - r_{comm,0} \times r_1 \times r_{comm,1}\right)^k \ ;$$
end
$F(0,0) = 1$;
for $i = 1$ **to** n **do**
  $F(i,0) = 0$;
end
for $i = 2$ **to** n **do**
  for $k = i$ **to** p **do**
    $$F(i,k) = \max_{1 \leq j < i, 1 \leq q \leq \min\{\mathcal{K},k\}} \left\{ F(j,k-q) \times \left(1 - \left(1 - r_{comm,j-1} \times \prod_{j \leq l \leq i} r_l \times r_{comm,i}\right)^q\right) \right\} ;$$
  end
end
$r = \max_{1 \leq q \leq p} F(n,q)$;

Theorem 3.1. *Algorithm 9 computes in time $O(n^2p^2)$ the optimal mapping for reliability optimization on fully homogeneous platforms.*

Proof. In this algorithm, $F(i,k)$ is the optimal reliability when mapping the first i tasks on k processors, and it is computed iteratively with the dynamic programming procedure. ■

3.5.2 Reliability/period optimization

We now present a bi-criteria (reliability, period) polynomial-time algorithm that optimizes the reliability of a mapping given a bound on the period. Recall that, for homogeneous platforms, the worst-case period and the expected period are the same.

Theorem 3.2. *Algorithm 10 computes in time $O(n^2p^2)$ the optimal mapping for reliability optimization on fully homogeneous platforms, when a bound on the period is given.*

Proof. In this algorithm, $F(i,k)$ is again the optimal reliability when mapping the first i tasks on k processors. The dynamic programming procedure of Algorithm 9 has been modified to account for the period bound. ■

Finally, we observe that the converse problem, namely optimizing the period when a bound on the reliability is enforced, is polynomial too. We can simply perform a binary search on the period and repeatedly execute Algorithm 10 until the optimal value is found.

3.5.3 Reliability/latency optimization

We now prove the NP-completeness of the bi-criteria (reliability, latency) optimization problem on homogeneous platforms. As for the period, there is no difference between the worst-case latency and the expected latency on such platforms.

Theorem 3.3. *The problem of optimizing the reliability on homogeneous platforms, with a bound on the latency, is NP-complete.*

Proof. Consider the associated decision problem: given a homogeneous platform, a chain of tasks, a bound $\mathcal{K}$ on the number of replications, a reliability r, and a latency L, does there exist a mapping whose reliability is at least r and whose latency is at most L? This problem is obviously in NP: given a mapping, it is easy to compute its reliability and latency, and to check that it is valid in polynomial time.

To establish the completeness, we use a reduction from 2-PARTITION (instance $\mathcal{I}_1$): given a set A of n numbers $a_1, \ldots, a_n$, does there exist a subset $A' \subset A$ such that $\sum_{a \in A'} a = \sum_{a \notin A'} a$. Let $T = \frac{1}{2}\sum_{a \in A} a$. Let $a_{min} = \min_{1 \leq i \leq n}\{a_i\}$ and $a_{max} = \max_{1 \leq i \leq n}\{a_i\}$. We build the following instance $\mathcal{I}_2$ of our problem with $3n+1$ tasks and $6n$ identical processors:

- $\mathcal{K} = 2$ and $\lambda = 10^{-8}10^{-n}a_{max}^{-3n}$;
- $s = b = 1$ (unit processor speed and link bandwidth);
- $B = \frac{1}{2a_{min}}\left(\frac{n}{4} + na_{max}^2 + T + 2\right)$;
- $\forall 1 \leq i \leq n$, $w_{3i-2} = B$, $w_{3i-1} = \frac{1}{2}$ and $w_{3i} = a_i$;
- $w_{3n+1} = B$;
- $\forall 1 \leq i \leq n, r_i = e^{-\lambda w_i}$ and $r_{comm,i} = 1$;
- $\forall 1 \leq i \leq n$, $o_{3i-2} = 0$, $o_{3i-1} = a_i$ and $o_{3i} = 0$;
- $L = (n+1)B + \frac{n}{2} + 3T$;

Algorithm 10: Optimal algorithm for reliability optimization on fully homogeneous platforms, when a bound on the period is given.

Data: a number p of fully homogeneous processors of failure rate λ, a list A of n tasks of sizes w_i, a maximal number $\mathcal{K}$ of replications, and an upper-bound P on the period

Result: a reliability r

for $k = 1$ **to** $\min\{\mathcal{K}, p\}$ **do**
  if $\max\left(\frac{o_0}{b}, \frac{w_1}{s}, \frac{o_1}{b}\right) \leq P$ **then**

$$F(1,k) = \left(1-(1-r_{comm,0}\times r_1 \times r_{comm,1})^k\right) \ ;$$

  else
    $F(1,k) = 0$;
  end
end
for $i = 1$ **to** n **do**
  $F(i,0) = 0$;
end
for *i=2* **to** *n* **do**
  for *k=i* **to** *p* **do**

$$F(i,k) = \max_{1\leq j<i, 1\leq q\leq \min\{\mathcal{K},k\}} \left\{ F(j,k-q)\times \left(1-\left(1-r_{comm,j}\times \prod_{j<l\leq i} r_l \times r_{comm,i}\right)^q\right) \right.$$
$$\left. |\max\left(\frac{o_j}{b}, \frac{\sum_{v=j+1}^{i} w_v}{s}, \frac{o_i}{b}\right) \leq P \right\} \ ;$$

  end
end
$r = \max_{1\leq q\leq p} F(n,q)$;

– it follows that the reliability of the mapping is

$$r = \left(1 - (1 - e^{-\lambda B})^2\right)^{n+1} \times$$

$$\left(1 - \lambda^2(\frac{n}{4} + \sum_{1 \leq i \leq n} a_i^2 + T) - \lambda^4 \times 2^{2n}(a_{max} + 1)^n\right) .$$

The size of instance $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. We now 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'. Then we propose the following solution for $\mathcal{I}_2$:

– all intervals are replicated 2 times;
– any task of size B make up an interval;
– for all $1 \leq i \leq n$, if $a_i \in A'$, then T_{3i-1} and T_{3i} are assigned to two different intervals, else they constitute one single interval.

This yields the following costs for the latency:

– the sum of computation costs does not depend of the mapping: $(n+1)B + \frac{n}{2} + 2T$;
– for each $a_i \in A'$, we add a communication cost a_i.

We thus obtain a latency $L = (n+1)B + \frac{n}{2} + 3T$. Concerning the reliability, it is the product of the reliability of all intervals:

– the reliability of intervals of size B is $(1-(1-e^{-\lambda B})^2)$;
– for each $a_i \in A'$, the product of the reliability of the two intervals for tasks T_{3i-1} and T_{3i-1} is $(1-(1-e^{-\frac{\lambda}{2}})^2)(1-(1-e^{-\lambda a_i})^2)$, which is greater than $(1-\frac{\lambda^2}{4})(1-\lambda^2 a_i^2)$;
– for each $a_i \notin A'$, the reliability of the interval for tasks T_{3i-1} and T_{3i-1} is $(1-(1-e^{-\lambda(a_i+\frac{1}{2})})^2)$, which is greater than $1-\lambda^2(a_i+\frac{1}{2})^2$.

We thus obtain, for the product of all these reliabilities,

$$\begin{aligned}
r' &= (1 - (1 - e^{-\lambda B})^2)^n \times \\
&\quad \textstyle\prod_{a_i \in A'}(1 - (1 - e^{-\frac{\lambda}{2}})^2)(1 - (1 - e^{-\lambda a_i})^2) \times \\
&\quad \textstyle\prod_{a-i \notin A'}\left(1 - \left(1 - e^{-\lambda\left(a_i + \frac{1}{2}\right)}\right)^2\right) \\
&\geq (1 - (1 - e^{-\lambda B})^2)^n \times \\
&\quad \textstyle\prod_{a_i \in A'}(1 - \frac{\lambda^2}{4})(1 - \lambda^2 a_i^2) \times \\
&\quad \textstyle\prod_{a-i \notin A'}(1 - \lambda^2(a_i + \frac{1}{2})^2) \\
&\geq (1 - (1 - e^{-\lambda B})^2)^n \times \\
&\quad \left(1 - \lambda^2\left(\frac{n}{4} + \textstyle\sum_{1 \leq i \leq n} a_i^2 + T\right) - \lambda^4 2^{2n}(a_{max} + 1)^n\right)
\end{aligned}$$

Suppose now that $\mathcal{I}_2$ has a solution. The exponent in the reliability bound implies that any interval is replicated at least 2 times, and the bound on replication is 2. This means that all intervals are replicated exactly 2 times. Suppose that one of the tasks of size B is computed together with another task in the

same interval. This yields the bound on reliability:

$$\begin{aligned} r' &< (1-(1-e^{-\lambda B})^2)^n(1-(1-e^{-\lambda(B+a_{min})})^2) \\ &< (1-(1-e^{-\lambda B})^2)^{n+1} \times \frac{1-\lambda^2(B+a_{min})^2}{1-\lambda^2 B^2(1-\frac{\lambda B}{2})^2} \\ &< (1-(1-e^{-\lambda B})^2)^{n+1}(1-\lambda^2(B+a_{min})^2) \\ &\quad (1+\lambda^2 B^2(1-\tfrac{\lambda B}{2})^2+2\lambda^4 B^4(1-\tfrac{\lambda B}{2})^4) \\ &< (1-(1-e^{-\lambda B})^2)^{n+1} \times (1-2\lambda^2 B a_{min}+7\lambda^4 B^4) \\ &< r \end{aligned}$$

This means that any task of size B makes up an interval. Let A' be the set of values i such that T_{3i-1} and T_{3i} are not in the same interval. We obtain the following formulas:

– For the reliability:

$$\begin{aligned} r &\leq (1-(1-e^{-\lambda B})^2)^n \times \\ &\quad \textstyle\prod_{a_i \in A'}(1-(1-e^{-\frac{\lambda}{2}})^2)(1-(1-e^{-\lambda a_i})^2) \times \\ &\quad \textstyle\prod_{a_i \notin A'}\left(1-\left(1-e^{-\lambda\left(a_i+\frac{1}{2}\right)}\right)^2\right) \\ &\leq (1-(1-e^{-\lambda B})^2)^n \times \\ &\quad \textstyle\prod_{a_i \in A'}(1-\frac{\lambda^2}{4}(1-\frac{\lambda}{4})^2)(1-\lambda^2 a_i^2(1-\lambda a_i)^2 \times \\ &\quad \textstyle\prod_{a_i \notin A'}(1-(\lambda^2+\frac{\lambda^2}{4}+\lambda^2 a_i)(1-\frac{\lambda}{2}(a_i+\frac{1}{2}))^2) \\ &\leq 1-\lambda^2(\tfrac{n}{4}+\textstyle\sum_{1\leq i\leq n} a_i^2+\sum_{a_i \notin A'} a_i)+\lambda^3 10^n a_{max}^{3n} \end{aligned}$$

– For the latency:

$$(n+1)B+\frac{n}{2}+\sum_{a_i \in A'} a_i + 2T \leq (n+1)B+\frac{n}{2}+3T$$

This means $\sum_{a_i \notin A'} a_i \leq T$ and $\sum_{a_i \in A'} a_i \leq T$. Hence, A' is a solution for $\mathcal{I}_1$. This concludes the proof. ■

We conclude that, on homogeneous platforms, the bi-criteria (reliability, period) problem is polynomial, while the bi-criteria (reliability, latency) problem is NP-complete. As a consequence, the tri-criteria (reliability, period, latency) problem is NP-complete too.

It is striking, and somewhat unexpected, that the bi-criteria (reliability, period) problem is easier than the (reliability, latency) one. The intuition for this difference is the following: when the period bound is given, we know once and for all which processors are fast enough to be enrolled for a given interval. Therefore, the mapping choices are local. On the contrary, the computation of the latency remains global, and its final value, including communication costs, depends upon the choices that will be made further on.

3.5.4 Integer linear program

In this section, we show how to derive an integer linear program (ILP) to solve the following problem: given an instance with n tasks and p homogeneous processors, bounds P on period and L on latency, compute the most reliable schedule respecting both bounds. Despite its high computation complexity, this ILP will be used on small problem instances to assess the absolute performance of the heuristics (see Section 3.8).

The ILP has $O(n^2 \times p)$ variables: for $1 \leq i \leq j \leq n$ and $1 \leq k \leq \min(p, \mathcal{K})$, $a_{i,j,k} = 1$ if the interval $\tau_i, ..., \tau_j$ is allocated onto k processors, and $a_{i,j,k} = 0$ otherwise. The objective function is the logarithm R of the reliability, which we want to maximize.

We list below the constraints that need to be enforced.

– Each task τ_i is included in exactly one interval:

$$\forall 1 \leq i \leq n, \sum_{1 \leq j \leq i} \sum_{i \leq k \leq n} \sum_{1 \leq \ell \leq p} a_{i,j,\ell} = 1 \ .$$

– At most p processors are used:

$$\sum_{1 \leq i \leq j \leq n} \sum_{1 \leq k \leq \mathcal{K}} k \times a_{i,j,k} \leq p \ .$$

– The latency bound is enforced:

$$\sum_{1 \leq i \leq j \leq n} \sum_{1 \leq k \leq \mathcal{K}} \frac{1}{s} (\sum_{i \leq \ell \leq j} w_\ell) \times a_{i,j,k} \leq L \ .$$

– The period bound is enforced:

$$\forall 1 \leq i, j \leq n, \frac{1}{s} (\sum_{i \leq \ell \leq j} w_\ell) \times \sum_{1 \leq k \leq \mathcal{K}} a_{i,j,k} \leq P \ ;$$

$$\forall 1 \leq i, j \leq n, o_j \times \sum_{1 \leq k \leq \mathcal{K}} a_{i,j,k} \leq P \ .$$

Finally, the objective function is to maximize the logarithm of the reliability R:

$$R = \sum_{1 \leq i \leq j \leq n} \sum_{k=1}^{\mathcal{K}} \log \left(1 - (1 - \exp^{\frac{\lambda}{s} \sum_{l=i}^{j} w_l})^k \right) \times a_{i,j,k} \ .$$

3.5.5 Allocation of intervals to processors

In this section, we consider that the partition into intervals is given, and we search for the best allocation of these intervals across the processors. This sub-problem is used in particular while designing heuristics in Section 3.7.

Once the intervals are fixed, since the platform is homogeneous, the period and latency are fixed. The allocation of processors only impacts the reliability. We derive below an optimal algorithm, ALGO-ALLOC, which assigns processors to intervals in order to maximize the reliability. The main idea is to allocate processors one by one to intervals, and the current interval is chosen so as to maximize the reliability.

The algorithm ALGO-ALLOC is the following:

– initially, we allocate one processor on each interval;

– then, while there remains an un-allocated processor and an interval replicated less than $\mathcal{K}$ times, we allocate a new processor on the interval whose ratio

$$\frac{\text{reliability with one more replica processor}}{\text{current reliability}}$$

is maximal.

We prove below the optimality of this algorithm.

Theorem 3.4. *Given a partition into intervals, Algorithm* ALGO-ALLOC *maximizes the reliability of the allocation.*

Proof. Let $I_1 \to \cdots \to I_i$ be the chain of intervals, where W_j (resp. o_j) is the computation cost (resp. communication cost) of interval I_j, for $1 \leq j \leq i$. Moreover, let k_j be the number of processors allocated to interval I_j by Algorithm ALGO-ALLOC, and let k'_j be the number of such processors in an optimal solution.

First, note that if $i \times \mathcal{K} \leq p$, ALGO-ALLOC allocates $\mathcal{K}$ processors per interval, and this is optimal, i.e., $\forall 1 \leq j \leq i$, $k_j = k'_j = \mathcal{K}$. Otherwise, all processors are allocated in both solutions, since it is always increasing reliability to replicate an interval once more, i.e., $\sum_{j=1}^{i} k_j = \sum_{j=1}^{i} k'_j = p$.

Indeed, for $2 \leq k \leq \mathcal{K}$, consider the increase in reliability when assigning a k-th processor to interval I_j:

$$R_{k,j} = \frac{\text{reliability of } I_j \text{ with } k \text{ processors}}{\text{reliability of } I_j \text{ with } k-1 \text{ processors}} .$$

We obtain $R_{k,j} = \frac{1-\alpha_j^k}{1-\alpha_j^{k-1}}$, with $\alpha_j = 1 - \exp^{-\lambda \frac{W_j}{s}}$. Note that this value is independent of the allocation of the other intervals. We derive:

$$\begin{aligned} R_{k+1,j} - R_{k,j} &= \frac{(1-\alpha_j^{k+1})(1-\alpha_j^{k-1}) - (1-\alpha_j^k)^2}{(1-\alpha_j^k)(1-\alpha_j^{k-1})} \\ &= \frac{2\alpha_j^k - \alpha_j^{k+1} - \alpha_j^{k-1}}{(1-\alpha_j^k)(1-\alpha_j^{k-1})} \\ &\leq 0 . \end{aligned}$$

by convexity of the function $x \to \alpha^x$. The ratio $R_{k,j}$ is thus decreasing with k.

Now suppose that there exist two values j_1 and j_2 with $k_{j_1} < k'_{j_1}$ and $k_{j_2} > k'_{j_2}$. To simplify notations, assume that $j_1 = 1$ and $j_2 = 2$, hence $k_1 < k'_1$ and $k_2 > k'_2$. Consider the iteration of ALGO-ALLOC during which the $(k'_2 + 1)$-th processor is added to interval I_2. At that point, there were $k^* \leq k_1$ processors assigned to I_1. By construction of ALGO-ALLOC, since interval I_2 is chosen, we have $R_{k'_2+1,2} \geq R_{k^*+1,1}$. Also, $R_{k^*+1,1} \geq R_{k'_1,1}$ because $k^* \leq k_1 < k'_1$ and $R_{k,1}$ is decreasing with k. We thus have $\frac{R_{k'_2+1,2}}{R_{k'_1,1}} \geq 1$: but this latter quantity is the variation of the global reliability when reassigning one processor from I_1 to I_2. Hence the allocation $(k'_1 - 1, k'_2 + 1, k'_3, ..., k'_i)$ is at least as reliable as the original optimal allocation $(k'_1, k'_2, k'_3, ..., k'_i)$, and therefore they are both optimal.

After a finite number of such reassignments, we obtain the allocation of ALGO-ALLOC, thereby establishing its optimality. ■

3.6 Complexity results for heterogeneous platforms

In this section, we prove the NP-completeness of the mono-criterion reliability optimization problem on heterogeneous platforms.

Theorem 3.5. *The problem of optimizing the reliability on heterogeneous platforms is NP-complete.*

Proof. Consider the associated decision problem: given a heterogeneous platform, a chain of tasks, a bound on the number $\mathcal{K}$ of replications, and a reliability r, does there exist a mapping of reliability at least r? This problem is obviously in NP: given a reliability and a mapping, it is easy to compute the reliability and to check that it is valid in polynomial time.

To establish the completeness, we use a reduction from 3-PARTITION. Consider the following general instance $\mathcal{I}_1$ of 3-PARTITION: given $3n$ numbers $a_1, \ldots, a_{3n}$ and a number T such that $\sum_{1 \leq j \leq 3n} a_j = nT$, does there exist n independent subsets $B_1, \ldots, B_n$ of $\{a_1, \ldots, a_{3n}\}$ such that for all $1 \leq i \leq n$, $\sum_{a_j \in B_i} a_j = T$?

We build the following instance $\mathcal{I}_2$ with n tasks and $p = 3n$ processors:
- $\lambda = \frac{10^{-8}}{nT^2}$;
- $\mathcal{K} = 3$;
- $\gamma = 1 + \frac{1}{2(T-1)}$;
- $\forall 1 \leq i \leq n, w_i = 1/n$ (all tasks have cost $1/n$);
- $r_{u,i} = e^{-\lambda_u \frac{w_i}{s_u}}$;
- $r_{comm,i} = 1$;
- $\forall 1 \leq u \leq 3n, \lambda_u = \lambda * \gamma^{a_u}$ and $s_u = 1$;
- it follows that the reliability of the mapping is $r = \left(1 - \lambda^3 \gamma^T\right)^n$.

The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. 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 $B_1, \ldots, B_n$. We propose the following solution for $\mathcal{I}_2$:
- we have one interval per task;
- the i-th task is replicated three times and allocated to the set of processors $\{P_u \mid u \in B_i\}$.

We obtain the following reliability for task i:

$$1 - \prod \left(1 - e^{-\lambda \gamma^{a_i}}\right) \geq 1 - \prod \left(\lambda \gamma^{a_i}\right) \geq 1 - \lambda^3 \gamma^T \ ,$$

Hence, the overall reliability is $r \geq (1 - \lambda^3 \gamma^T)^n$.

Suppose now that $\mathcal{I}_2$ has a solution. We first show that the optimal mapping consists of n intervals, one per task, each replicated three times. Suppose that we know the number of intervals in the optimal mapping. There are at most n intervals, and we have enough processors to replicate all of them three times, and this increases the reliability. We conclude that all intervals will be replicated three times. Suppose now that one of these intervals contains $t > 1$ tasks. There are enough processors to split this interval into t single-task intervals, each replicated three times. Let r_1 be the reliability of the original interval with t tasks, and r_t the reliability of the same tasks assigned to t intervals replicated three times. By hypothesis of optimality, we have:

$$\begin{array}{lllll}
 & r_1 & \geq & r_t & \\
\Rightarrow & e^{-\lambda\gamma t} & \geq & 1 - (1 - e^{-\lambda\gamma^T})^t & \\
\Rightarrow & \lambda\gamma t - \frac{1}{2}(\lambda\gamma t)^2 & \leq & (\lambda\gamma^T)^t & \text{because } \lambda\gamma^T \leq 1 \\
\Rightarrow & \lambda\gamma 2 - \frac{1}{2}(\lambda\gamma 2)^2 & \leq & (\lambda\gamma 2)^2 & \text{because } \gamma^{T-1} \leq 2 \\
\Rightarrow & \lambda\gamma 2 & \leq & \frac{3}{2}(\lambda\gamma 2)^2 & \\
\Rightarrow & \lambda\gamma 2 & \geq & \frac{2}{3} & \\
\Rightarrow & 4\lambda & \geq & \frac{2}{3} &
\end{array}$$

However, $\lambda \leq 10^{-8}$, which contradicts the hypothesis. This means that, in the optimal solution, any task constitutes an interval.

Let, for all i, $B_i = \{a_j \mid T_i \text{ mapped on } P_j\}$. We obtain the following reliability:

$$r = \prod_{1 \leq i \leq n} (1 - \prod_{a_j \in B_i} (1 - e^{-\lambda \gamma^{a_i}})) \geq (1 - \lambda^3 \gamma^T)^n.$$

Suppose that, for a value i, $\sum_{a_j \in B_i} a_j \neq T$. Then,

$$\begin{aligned}
r &\leq \textstyle\prod_{1 \leq i \leq n} (1 - \prod_{a_j \in B_i} (\lambda \gamma^{a_i} - \frac{1}{2} (\lambda \gamma^{a_i})^2)) \\
&\leq \textstyle\prod_{1 \leq i \leq n} (1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j} \prod_{a_j \in B_i} (1 - \frac{1}{2} \lambda \gamma^{a_i})) \\
&\leq \textstyle\prod_{1 \leq i \leq n} (1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j} (1 - \frac{\lambda}{2} \sum_{a_j \in B_i} \gamma^{a_j})) \\
&\leq \textstyle\prod_{1 \leq i \leq n} (1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j} (1 - \frac{3\lambda}{2} \gamma^T)) \\
&\leq \textstyle\prod_{1 \leq i \leq n} (1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j} + \frac{3\lambda^4}{2} \gamma^{T + \sum_{a_j \in B_i} a_j}) \\
&\leq \textstyle\prod_{1 \leq i \leq n} (1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j}) (1 + \frac{\frac{3\lambda^4}{2} \gamma^{T + \sum_{a_j \in B_i} a_j}}{1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j}}) \\
&\leq \textstyle\prod_{1 \leq i \leq n} (1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j}) (1 + \frac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}}) \\
&\leq (1 + \tfrac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}})^n \times \textstyle\prod_{1 \leq i \leq n} (1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j})
\end{aligned}$$

By hypothesis, we have $\sum_{a_j \in B_i} a_j \neq T$ for a value i. Then by convexity,

$$\prod_{1 \leq i \leq n} (1 - \lambda^3 \gamma^{\sum_{a_j \in B_i} a_j}) \leq (1 - \lambda^3 \gamma^T)^{n-2} \times (1 - \lambda^3 \gamma^{T-1}) \times (1 - \lambda^3 \gamma^{T+1}) \, .$$

By hypothesis, we have:

$$\begin{aligned}
(1 - \lambda^3 \gamma^T)^n &\leq r \\
&\leq (1 + \tfrac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}})^n (1 - \lambda^3 \gamma^T)^{n-2} \\
&\quad (1 - \lambda^3 \gamma^{T-1})(1 - \lambda^3 \gamma^{T+1}) \\
\Rightarrow (1 - \lambda^3 \gamma^T)^2 &\leq (1 + \tfrac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}})^n \\
&\quad (1 - \lambda^3 \gamma^{T-1})(1 - \lambda^3 \gamma^{T+1}) \\
&\leq (1 + \tfrac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}})^n \\
&\quad \left((1 - \lambda^3 \gamma^T)^2 - \lambda^3 \gamma^{T-1} (\gamma - 1)^2\right) \\
\Rightarrow (1 - \lambda^3 \gamma^T)^2 \times \left((1 + \tfrac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}})^n - 1\right) &\geq \left((1 + \tfrac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}})^n - 1\right) \times \left(\lambda^3 \gamma^{T-1} (\gamma - 1)^2\right) \\
\Rightarrow (1 - \lambda^3 \gamma^T)^2 &\geq \left((1 + \tfrac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}})^n - 1\right)^{-1} \\
&\quad (1 + \tfrac{\frac{3\lambda^4}{2} \gamma^{4T}}{1 - \lambda^3 \gamma^{3T}})^n \lambda^3 \gamma^{T-1} (\gamma - 1)^2 \\
&\geq \tfrac{1 + \frac{3\lambda^4}{4} n \gamma^{4T}}{\frac{3\lambda^4}{4} n \gamma^{4T}} \lambda^3 \gamma^{T-1} (\gamma - 1)^2 \\
&\geq \tfrac{1 + \frac{3\lambda^4}{4} n \gamma^{4T}}{3\lambda n \gamma^{3T+1} (T-1)^2}
\end{aligned}$$

However, $3\lambda n \gamma^{3T+1} (T-1)^2 \leq 1$ and $1 + \frac{3\lambda^4}{4} n \gamma^{4T} \geq 1$. This contradicts the hypothesis. Then, if $\{B_1, \ldots, B_n\}$ corresponds to a solution of $\mathcal{I}_2$, we have $\sum_{a_j \in B_i} a_j = T$ for $1 \leq i \leq n$. This shows that $B_1, \ldots, B_n$ is a solution for $\mathcal{I}_1$, which concludes the proof. ■

Because mono-criterion reliability optimization is already NP-complete, all multi-criteria problems with period or latency or both, are also NP-complete on heterogeneous platforms.

3.7 Heuristics

In this section, we present two heuristics to compute schedules for the multi-criteria problem discussed above. Since we consider several criteria, each heuristic algorithm returns, for a given problem instance, several possible schedules. In the experiments of Section 3.8, both the period and the latency are bounded and, for each instance and each heuristic, we select, in the set of computed solutions, the schedule having the best reliability while still meeting the bounds on period and latency.

Each heuristic consists of two steps: in a first step, the chain of tasks is divided into intervals, and in the second step, the processors are allocated to these intervals. We present the algorithms used in the two steps by the two heuristics.

3.7.1 Computation of the intervals

We consider two possible ways to compute the intervals. In both cases, we first decide the number of intervals used, and then we compute intervals according to this value. We can thus compute a set of intervals for any possible number of intervals.

In the first heuristic, we try to minimize the latency. Thus, for i intervals, we select the intervals yielding the $i-1$ smallest communication costs. More precisely, for i intervals, we consider the output communication costs of all tasks except the last one. Let $u_1 < \cdots < u_{i-1}$ be the $i-1$ smallest communication costs. Then, the first interval contains tasks τ_1 to τ_{u_1}, the second interval contains tasks τ_{u_1+1} to τ_{u_2}, and so on; the last interval contains tasks $\tau_{u_{i-1}+1}$ to τ_n. This heuristic, denoted HEUR-L, is presented in Algorithm 11.

Algorithm 11: Heuristic HEUR-L for the computation of the intervals.

Data: n tasks of sizes w_i and of output communication cost o_i, a maximal number $\mathcal{K}$ of replications and a number i of intervals
Result: a set of intervals
Sort in array A the $n-1$ first tasks in increasing order of output communication cost;
Sort the $i-1$th first tasks of A in increasing order of placement in the chain;
The first interval contains tasks from τ_1 to $\tau_{A[1]}$;
for $j = 2$ **to** $i-1$ **do**
  The jth interval contains tasks from $\tau_{A[j-1]+1}$ to $\tau_{A[j]}$;
end
The last interval contains tasks from $\tau_{A[i-1]+1}$ to τ_n;

In the second heuristic, we try to minimize the period. We thus try to obtain intervals which are as identical as possible in size. We use a dynamic programming algorithm to compute the optimal period in the homogeneous case. More precisely, let $F(j,k)$ be the optimal period that can be obtained by grouping the j first tasks into k intervals. The initialization is $F(j,1) = \max\{\sum_{l \leq j} w_l, o_j\}$, and the recurrence writes:

$\forall k \geq 2, \forall j \leq k,$

$$F(j,k) = \min_{1 \leq j' < j} \left\{ \max \left(F(j', k-1), \sum_{j' < l \leq j} w_l,\ o_j \right) \right\}.$$

This heuristic, denoted HEUR-P, is presented in Algorithm 12.

Algorithm 12: Heuristic HEUR-P for the computation of the intervals.

Data: n tasks of size w_i and of output communication cost o_i, a maximal number $\mathcal{K}$ of replications and a number i of intervals
Result: a set of intervals
for $j = 1$ **to** n **do**
  $F(j,1) = (\max\{\sum_{l \leq j} w_l, o_j\}, 1)$;
end
for $k = 1$ **to** i **do**
  for $j = 1$ **to** n **do**
    Let j' be the value that minimize the function
    $x \to \max\left\{fst(F(x,k-1)), \sum_{x<l\leq j} w_l, o_j\right\}$;
    $F(j,k) = \left(\max\left\{fst(F(j',k-1)), \sum_{j'<l\leq j} w_l, o_j\right\}, j'\right)$;
  end
end
Let I be an array of size i;
$j = n$;
$k = i$;
while $j \geq 1$ **do**
  $I[k-1] = j$;
  $j \leftarrow snd(F(j,k))$;
end
for $j = 1$ **to** $i-1$ **do**
  The jth interval contains tasks $\tau_{I[j-1]+1}$ to $\tau_{I[j]}$;
end
The last interval contains tasks $\tau_{I[i-1]+1}$ to τ_n;

Both heuristics produce $\min\{n, p\}$ possible divisions in intervals of the chain of tasks. It remains to allocate processors to these intervals. This is presented in the next section.

3.7.2 Allocation of processors to intervals

As presented in Section 3.5.5, Algorithm ALGO-ALLOC allocates processors optimally to intervals in the homogeneous case. We use a variant of algorithm ALGO-ALLOC in the general case with a bound P on the period: at the beginning of the algorithm, in increasing order of value $\frac{\lambda_u}{s_u}$, a processor is allocated to the longest possible interval that has no processor allocated to it. Then, step by step, on the remaining processors, processor P_u is allocated to the interval of greatest value $\left(\frac{\text{reliability with this processor}}{\text{reliability at current step}}\right)$ among the intervals I_j such that $\frac{W_j}{s_u} \leq P$. This corresponds to algorithm ALGO-ALLOC: we first allocate the more reliable processors, and we do not allocate a processor to an interval if the associated computation time exceed the bound on period.

3.8 Experiments

This section reports experimental results assessing the performance of the heuristics HEUR-P and HEUR-L. In the homogeneous case, these heuristics are compared with the optimal solution computed with the integer linear program presented in Section 3.5.4. The heuristics are developed in C/C++ and the integer linear program is implemented with CPLEX [20]. The reader can find the corresponding source code at [26].

3.8.1 Experiments on homogeneous platforms

To measure the performance of the HEUR-P and HEUR-L heuristics, we randomly generated workflow applications. For 15 tasks and 10 processors, we randomly generated values of communication and computation costs for 100 different chains of tasks. Then, for any reasonnable bounds on period and latency, we compute, for the integer linear program and for both heuristics, the number of solutions found within these bounds, and the average ratio of the optimal reliability (integer linear program) and the reliability of the heuristics; we count 0 for instances without any solution.

We set the processor speeds to $s = 1$ computation per time unit, and computation costs of tasks are randomly chosen in the interval $[1, 100]$. The bandwidth is set to $b = 1$, and communication costs are randomly chosen in the interval $[1, 10]$. The failure rate per time unit of processors (resp. communication links) are set to $\lambda_p = 10^{-6}$ (resp. $\lambda_\ell = 10^{-5}$) per time unit. These values are realistic for modern fail-silent hardware components, where the unit of time is the hour [8].

With these values, 100 problem instances are generated with 10 processors and a chain of 15 tasks. The maximum number of replication is fixed to $\mathcal{K} = 3$. The reasonable period values are found between 70 and 140 time units, and latency between 500 and 1000 time units.

In Figure 3.6, the latency is bounded by 750, and the bound on period is chosen in the interval $[50, 500]$. For higher values of period, neither the ILP program nor the heuristics are able to solve the problem instance. The value selected for the bound on the latency corresponds to the minimum value such that approximately half of the instances can be solved when there are no constraints on the period. The HEUR-P heuristic finds solutions for most of the instances which have a solution, except for average values of period ($100 \leq P \leq 200$), and for high values ($P > 400$). Concerning the HEUR-L heuristic, it finds fewer solutions than HEUR-P for low and average values of the period ($P \leq 400$), but it obtains more results than HEUR-P and as many as the ILP program for high values of the period.

To summarize these results, the HEUR-P heuristic obtains good results in most cases, even though it does not consider latency, hence leading to poorer results when the period is not constrained at all. However, the HEUR-L heuristic becomes efficient for high values of the period, since it focuses on the latency criterion. In contrast, HEUR-L seems to be less efficient than HEUR-P, since it obtains fewer results for reasonable problem instances with a bound on the period.

In Figure 3.7, we bound the period by $P = 250$, while the bound on the latency is chosen in the interval $[500, 1100]$. The reason is that no solutions are found when either $L < 500$ or $L > 1100$. In this case, almost all existing solutions are found by both heuristics for low values of latency. For higher values of latency, HEUR-P remains efficient, while HEUR-L becomes less efficient. Even without any bound on the latency, HEUR-L fails to find some of the solutions. This can be explained by the fact that there are more tasks than processors in the instances that we consider (15 tasks and 10 processors). Since HEUR-L does not consider the size of intervals (but only the cost of communications), for some instances, even with the maximum number of intervals (10 with the instances considered), it can happen that an interval is too large and exceeds the bound on the period.

In Figures 3.6 and 3.7, we have compared the performances of both heuristics HEUR-L and HEUR-P

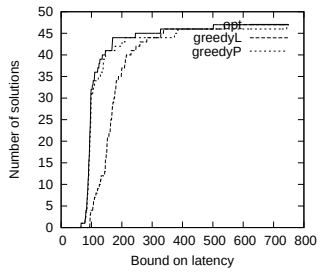

Figure 3.6: Number of solutions for $L = 750$.

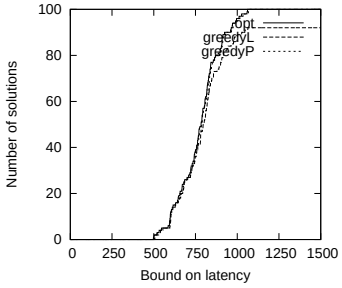

Figure 3.7: Number of solutions for $P = 250$.

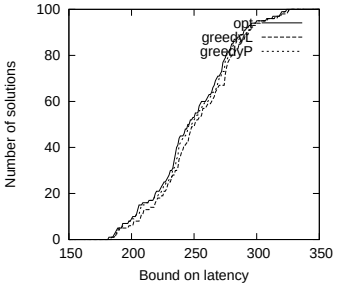

Figure 3.8: Number of solutions for $L = 3P$.

for any bound on the period with an average fixed latency, and any bound on the latency with an average fixed period. In the last experiment, we consider the case of a linear relationship between the value of period and the value of latency. In Figure 3.8, the period is taken in interval $[150, 350]$, and we fix the latency to be $L = 3P$. In this case, almost all solutions are found by both heuristics, whatever the bound on period. Note that in most cases, HEUR-P is slightly more efficient than HEUR-L, which confirms our previous observations.

We have not presented reliability results in this experiment, but we focused on the number of solutions found, given a period and a latency. Indeed, the low number of tasks makes the computation very reliable for any schedule: the reliability is always either very close to one, or equal to zero when no solution is found. In practice, experimental results give a reliability of 1 in type double, which means that the real value of the reliability is equal to 1 with an error of the order of 10^{-15}.

3.8.2 Experiments on heterogeneous platforms

On heterogeneous platforms, we were no longer able to use the ILP to compute the optimal solution. However, we have performed several experiments with a larger number of tasks and processors: we generated one random application with 50 tasks and 75 processors. The processor speeds were randomly chosen in the interval $[1, 100]$, while their failure rates per time unit where randomly chosen between 10^{-6} and 10^{-8}.

We have computed the optimal solutions of heuristics HEUR-P and HEUR-L for any reasonable values for the bounds on period and latency: the period was chosen between 50 and 400 time units, while the latency was chosen between 3600 and 5000 time units. Figure 3.9 plots in 3D the values of reliability computed for such values of period and latency, for both heuristics. For an enhanced readability, Figure 3.10 details the results obtained by HEUR-P alone, while Figure 3.11 focuses on HEUR-L alone.

From these figures, we first note that the bound on the latency does not impact the result. However, when the bound on the period is increased, we can find more reliable schedules. Once again, the supremacy of HEUR-P is demonstrated: not only does this heuristic find more solutions than HEUR-L, but it also finds more reliable schedules.

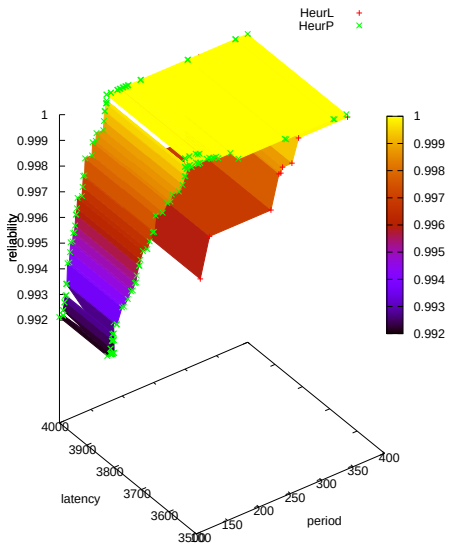

Figure 3.9: Reliability in the heterogeneous case.

3.9 Conclusion

We have addressed problems related to the mapping of linear workflows on homogeneous and heterogeneous distributed platforms. The main goal was to optimize the reliability of the mapping through task replication, while enforcing bounds on performance-oriented criteria (period and latency). We derived a comprehensive set of NP-hardness complexity results, together with optimal algorithms for polynomial instances. Altogether, these results provide a solid theoretical foundation for the study of multi-criteria mappings of linear workflows. Another contribution of this chapter is the introduction of a realistic communication model that nicely accounts for the inherent physical limitations on the communication capabilities of state-of-the-art processors.

Communication failures have been incorporated in the model through routing operations, which guarantee that evaluating the system reliability remains computationally tractable. An interesting future research direction would be to investigate whether it is feasible to remove this routing procedure, and accurately approximate the reliability of general systems (non serial-parallel).

On homogeneous platforms, an integer linear program is presented to solve the problem of maximizing the reliability with bounds on period and on latency, while polynomial-time heuristics are derived for the most general problems. We have proposed two heuristics: HEUR-L that attempts to minimize the latency and HEUR-P that attempts to minimize the period. Our experiments demonstrate the efficiency of the heuristics, and the supremacy of HEUR-P in most cases.

Another direction for future work involves the design of heuristics for even more difficult problems that would mix performance-related criteria (period, latency) with several other objectives, such as reliability, resource costs, and power consumption.

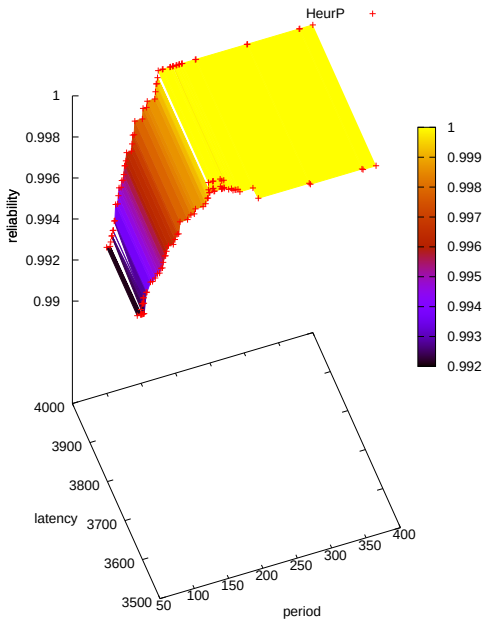

Figure 3.10: Reliability in the heterogeneous case for HEUR-P.

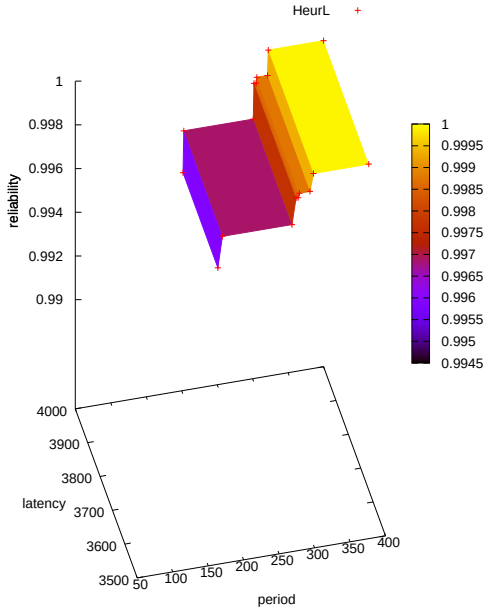

Figure 3.11: Reliability in the heterogeneous case for HEUR-L.

Chapter 4

Scheduling parallel iterative applications on volatile resources

4.1 Introduction

In the previous chapter, we studied a reliability problem, considering only transient failures. Even if fail-stop failures are less common than transient failures on classical grids, fail-stop failures are an important issue in some special models of platforms. The desktop grids are particularly sensible to such failures.

We study the problem of efficiently executing parallel applications on platforms that comprise volatile resources. More specifically we focus on iterative applications implemented using the master-worker paradigm. The master coordinates the computation of each iteration as the execution of a fixed number of independent tasks. A synchronization of all tasks occurs at the end of each iteration. This scheme applies to a broad spectrum of scientific computations including, but not limited to, mesh based solvers (e.g., elliptic PDE solvers), signal processing applications (e.g., recursive convolution), and image processing algorithms (e.g., stencil algorithms). We study such applications when they are executed on networked processors whose availability evolves over time, meaning that each processor alternates between being available for executing a task and being unavailable.

Solutions for executing master-worker applications, and in particular applications implemented with the Message Passing Interface (MPI), on failure-prone platforms have been developed (e.g., [29, 23, 54, 14]). In these works, the focus is on tolerating *failures* caused by software or hardware faults. For instance, a software fault will cause the processor to stall, but computations may be restarted from scratch or be resumed from a saved state after rebooting. A hardware failure may keep the processor down for a long period of time, until the failed component is repaired or replaced. In both cases, fault-tolerant mechanisms are implemented in the aforementioned solutions to make faults transparent to the application execution.

In addition to failures, processor volatility can also be due to *temporary interruptions*. Such interruptions are common in volunteer computing platforms [13] and desktop grids [19]. In these platforms processors are contributed by resource owners who can reclaim them at any time, without notice, and for arbitrary durations. A task running on a reclaimed processor is simply suspended. At a later date, when the processor is released by its owner, the task can be resumed without any wasted computation. In fact, fault-tolerant MPI solutions were proposed in the specific context of desktop grids [14], to accommodate for such interruptions. While mechanisms for executing master-worker applications on volatile platforms are available, our focus is on scheduling algorithms for deciding which processors should run which tasks and when.

At a given time a (volatile) processor can be in one of three states: UP (available), $DOWN$ (crashed due to a software or hardware fault), or $RECLAIMED$ (temporarily preempted by owner). Accounting for the $RECLAIMED$ state, which arises in desktop grid platforms, complexifies scheduling decisions. More specifically, since before going to the $DOWN$ state a processor may alternate between the UP and $RECLAIMED$ states, the time needed by the processor to compute a given workload to completion is difficult to predict. A way to make such prediction tractable is to assume that state transitions obey a Markov process. The Markov (i.e., memoryless) assumption is popular because it enables analytical derivations. In fact, recent work on desktop grid scheduling has made use of this assumption [17]. Unfortunately, the memoryless assumption is known not to hold in practice. Several authors have reported that the durations of availability intervals in production desktop grids are not sampled from exponential distributions [59, 82, 49]. There is no true consensus regarding what is a "good" model for availability intervals defined by the elapsed time between processor *failures*, let alone regarding a model for the durations of *recoverable interruptions*. Note that some authors have attempted to model processor availabilities using (non-memoryless) semi-Markov processes [64]. Faced with the lack of a good model for transitions between the UP, $DOWN$, and $RECLAIMED$ states, and not knowing whether such a model would be tractable or not, for now we opt for the Markovian model. The goal of this work is to provide algorithmic foundations for scheduling iterative master-worker applications on processors that can fail or be temporarily reclaimed. A 3-state Markovian model allows us to achieve this goal, and the insight from our results should provide guidance for dealing with more complex, and hopefully more realistic, stochastic models of processor availabilities. The transition probabilities for a real-life platform can be determined using traces. For example, for a given processor P_q, knowing that the ressource is UP at time t, the probability that it remains UP will be computed as the number of occurences of two successives UP states, divided by the number of UP states during all time slots described by the trace of this ressource.

A unique aspect of this work is that we account for network bandwidth constraints for communication between the master and the workers. More specifically, we bound the total outgoing communication bandwidth of the master while ensuring that each communication uses a reasonably large fraction of this bandwidth. The master is thus able to communicate simultaneously with only a limited number of workers, sending them either the application program or input data for tasks. This assumption, which corresponds to the bounded multi-port model [48], applies to concurrent data transfers implemented with multi-threading. One alternative is to simply not consider these constraints. In this case, a scheduling strategy could enroll a large (and vastly suboptimal) number of processors to which it would send data concurrently each at very low bandwidth. Another alternative is to disallow concurrent data transfers from the master to the workers. In this case, the bandwidth capacity of the master may not be fully exploited, especially for workers executing on distant processors. We conclude that considering the above bandwidth constraints is necessary for applications that do not have extremely low communication-to-computation ratios. It turns out that the addition of these constraints makes the problem dramatically more difficult at the theoretical level, and thus complicates the design of practical scheduling strategies.

The specific scheduling problem under consideration is to maximize the number of application iterations that are successfully completed before a deadline. Informally, during each iteration, we have to identify the "best" processors among those that are available (e.g., the fastest, the likeliest to remain available, etc.). In addition, since processors can become available again after being unavailable for some time, it may be beneficial to change the set of enrolled processors even if all enrolled processors are available. We thus have to decide whether to release enrolled processors, to decide which ones should be released, and to decide which ones should be enrolled instead. Such changes come at a price: the application program file must be sent to newly enrolled processors, which consumes some (potentially precious) fraction of the master's bandwidth.

Our contributions are the following. First, we assess the complexity of the problem in its off-line version, i.e., when processor availability behaviors are known in advance. Even with this knowledge, the problem is NP-hard, and cannot be approximated within a factor $8/7$. Next, relying on the Markov assumption for processor availability, we provide a closed-form formula for the expectation of the time needed by a worker to complete a set of tasks. This formula is at the heart of several heuristics that aim at giving priority to "reliable" resources rather than to "fast" ones. In a nutshell, when the task size is very small in comparison to the expected duration of an interval between two consecutive processor state changes, "classical" heuristics based upon the estimated completion time of a task perform reasonably well. But when the task size is no longer negligible with respect to the expected duration of such an interval, it is mandatory to account for processor reliability, and only those heuristics building upon such knowledge are shown to achieve good performance. Altogether, we design a set of heuristics, which we thoroughly evaluate in simulation. The results provide insights for selecting the best strategy as a function of processor state availability versus task duration.

This chapter is organized as follows. Section 4.2 discusses related work. Section 4.3 describes the application and platform models. Complexity results for the off-line study are given in Section 4.4; these results do not rely on any assumption regarding stochastic distribution of resource availability. In Section 4.5, we describe our 3-state Markovian model of processor availability, and we show how to compute the expected time for a processor to complete a given workload. Heuristics for the on-line problem are described in Section 4.6, some of which use the result in Section 4.5 for more informed resource selection. An experimental evaluation of the heuristics is presented in Section 4.7. Section 4.8 concludes with a summary of our findings and perspectives on future work.

4.2 Related work

There is a large literature on scheduling master-worker applications, or applications that consist of a sequence of iterations where each iteration can be executed in master-worker fashion [6, 44, 55]. In this work we focus on executions on volatile resources, such as desktop resources. The volatility of desktop or other resources is well documented and characterizations have been proposed [59, 82, 49]. Several authors have studied the master-worker (or "bag-of-tasks") scheduling problem in the face of such volatility in the context of desktop grid computing, either at an Internet-wide scale or within an Enterprise [53, 89, 27, 2, 17, 28, 81, 45]. Most of these works propose simple greedy scheduling algorithms that rely on mechanisms to pick processors according to some criteria. These processor selection criteria include static ones (e.g., processor clock-rates or benchmark results), simple ones based on past host behavior [53, 27, 28], and more sophisticated ones based on statistical analysis of past host availability [81, 45, 2, 17]. In a global setting, the work in [89] includes time-zone as a criterion for processor selection. These criteria are used to rank processors, but also to exclude them from consideration [53, 27]. The work in [17] is particularly related to our own in that it uses a Markov model of processor availability (but without accounting for preemption). Most of these works also advocate for task replication as a way to cope with volatile resources. Expectedly, injecting task replicas is sensible toward the end of application execution. Given the number of possible variants of scheduling algorithms, in [28] the authors propose a method to automatically instantiate the parameters that together define the behavior of a scheduling algorithm. Works published in this area are of a pragmatic nature, and few theoretical results have been sought or obtained (one exception is the work in [32]).

A key difference between our work and all the above is that we seek to develop scheduling algorithms that explicitly manage the master's bandwidth. Limited master bandwidth is a known issue for desktop grid computing [58, 76, 43] and must therefore be addressed even though it complexifies the scheduling

problem. To the best of our knowledge no previous work has made such an attempt.

4.3 Problem Definition

In this section, we detail our application and platform models, describe the scheduling model, and provide a precise statement of the scheduling problem.

4.3.1 Application Model

We target an iterative application in which iterations entail the execution of a fixed number m of same-size independent tasks. Each iteration is executed in a master-worker fashion, with a synchronization of all tasks at the end of the iteration. A processor is assigned one or more tasks during an iteration. Each task needs some input data, of constant size V_{data} in bytes. This data depends on the task and the iteration, and is received from the master. Such applications allow for a natural overlap of computation and communication: computing for the current task can occur while data for the next task (of the same iteration) is being received. Before it can start computing, a processor needs to receive the application program from the master, which is of size V_{prog} in bytes. This program is the same for all tasks and iterations.

4.3.2 Platform Model

We consider a platform that consists of p *processors*, $P_1, \ldots, P_p$, encompassing with this term compute nodes that contain multiple physical processor cores. Each processor is volatile, meaning that its availability for computing application tasks varies over time. More precisely, a processor can be in one of three states: UP (available for computation), $RECLAIMED$ (temporarily reclaimed by its owner), or $DOWN$ (crashed and to be rebooted). We assume that the master, which implements the scheduling algorithm, executes on a host that is always UP (otherwise a simple redundancy mechanism such as primary back-up [39] can be used to ensure reliability of the master). We also assume that the master is aware of the states of the processors, e.g., via a simple heart-beat mechanism [72]. Processor availabilities evolve independently, and all state transitions are allowed, with the following implications:

- When a UP or $RECLAIMED$ processor becomes $DOWN$, it loses the application program, all the data for its assigned tasks, and all partially computed results. When it later becomes UP it has to acquire the program again before executing tasks;
- When a UP processor becomes $RECLAIMED$, its activities are suspended. However, when it becomes UP again it can simply resume task computations and data transfers.

We discretize time so that the execution occurs over a sequence of discrete *time slots*. We assume that task computations and data transfers all require an integer number of time slots, and that processor state changes occur at time-slot boundaries. We leave the time slot duration unspecified. The time slot duration that achieves a good approximation of continuous time varies for different applications and platforms.

The temporal availability of P_q is described by a vector $\mathcal{S}_q$ whose component $\mathcal{S}_q[t] \in \{u, r, d\}$ represents its state at time-slot t. Here u corresponds to the UP state, r to the $RECLAIMED$ state, and d to the $DOWN$ state. Vector $\mathcal{S}_q$ is unknown before executing the application.

Processor P_q requires w_q time-slots of availability (i.e., UP state) to compute a task. If all w_q values are identical, then the platform is homogeneous. We model communications between the master and the workers using the *bounded multi-port* communication model [48]. In this model, the master can initiate multiple concurrent communications, each to a different worker. Each communication is allotted

a bandwidth fraction of the master's network card, and the sum of all fractions cannot exceed the total capacity of the card. This model is enabled by popular multi-threaded communication libraries [38]. We consider that the master can communicate up to bandwidth BW (we use the term "bandwidth" loosely to mean maximum data transfer rate). Communication to each worker is performed at some fixed bandwidth bw. This bandwidth can be enforced in software or can correspond to same-capacity communication paths from the master's processor to each other processor. We define $n_{com} = \text{BW}/\text{bw}$ as the maximum number of workers to which the master can send data simultaneously (i.e., the maximum number of simultaneous communications). For simplicity, we assume n_{com} to be an integer. Let n_{prog} be the number of processors receiving the application program at time t, and n_{data} be the number of processors receiving the input data of a task at time t. Given that the bandwidth of the master must not be exceeded, we have

$$n_{\text{prog}} + n_{\text{data}} \leq n_{com} = \text{BW}/\text{bw}.$$

Let P_q be a processor engaged in communication at time t, for receiving either the program or input data. In both cases, it does this with bandwidth bw. Hence the time for a worker to receive the program is $T_{\text{prog}} = V_{\text{prog}}/\text{bw}$, and the time to receive the data is $T_{\text{data}} = V_{\text{data}}/\text{bw}$.

4.3.3 Scheduling Model

Let $\mathit{config}(t)$ denote the set of processors enrolled for computing the m application tasks in an iteration, or *configuration*, at time t. Enrolled processors work independently, and execute their tasks sequentially. While a processor could conceivably execute two tasks in parallel (provided there is enough available memory), this would only delay the completion time of the first task, thereby increasing the risk of not completing it at all due to volatile availability. The scheduler assigns tasks to processors and may choose a new configuration at each time-slot t. Let P_q be a newly enrolled processor at time t, i.e., $P_q \in \mathit{config}(t+1) \setminus \mathit{config}(t)$. P_q needs to receive the program unless it already received a copy of it and has not been in the $DOWN$ state since. In all cases, P_q needs to receive data for a task before computing it. This holds true even if P_q had been enrolled at some previous time-slot $t' < t$ but has been un-enrolled since: we assume any received data is discarded when a processor is un-enrolled. In other words, any input data communication is resumed from scratch, even if it had previously completed. Note that a processor that is un-enrolled keeps the application program until it eventually goes to the $DOWN$ state.

If a processor becomes $DOWN$ at time t, the scheduler may simply use the remaining UP processors in $\mathit{config}(t)$ to complete the iteration, or enroll a new processor. Even if all processors in $\mathit{config}(t)$ are in the UP state, the scheduler may decide to change the configuration. This can be useful if a more desirable (e.g., faster, more available) but un-enrolled processor has just returned to the UP state. Removing an UP processor from $\mathit{config}(t)$ has a cost: partial results of task computations, partial task data being received, and previously received task data are all lost. Note, however, that results obtained for previously completed tasks are not lost because already sent back to the master. Due to the possibility of a processor leaving the configuration (either due to becoming $DOWN$ or due to a decision of the scheduler), the scheduler enforces that task data is received for at most one task beyond the one currently being computed. In other terms, the processor does not accumulate task data beyond that for the next task. This is sensible so as to allow some overlap of computation and communication while avoiding wasting bandwidth for data transfers that would be increasingly likely to be redone from scratch.

4.3.4 Problem Statement

The scheduling problem we address in this work is that of maximizing the number of successfully completed application iterations before a deadline. Given the discretization of time, the objective of the scheduling problem is then to maximize the number of successfully completed iterations within some integral number of time slots, N. In the off-line case (see Section 4.4), if an efficient algorithm can be found to solve this problem, then, using a binary search, an efficient algorithm can be designed to solve the problem of executing a given number of iterations in the minimum amount of time.

4.4 Off-line complexity

In this section, we study the off-line complexity of the problem. This means that we assume a priori knowledge of all processor states. In other words, the value of $\mathcal{S}_q[j]$ is known in advance, for $1 \leq q \leq p$ and $1 \leq j \leq N$. The problem turns out to be difficult: even minimizing the time to complete the first iteration with same-speed processors is NP-complete. We also identify a polynomial instance with $n_{com} = +\infty$, which highlights the impact of communication contention. For approximation questions, we take into account incomplete instances: if p tasks have been executed in the current time slot, then for the current iteration we consider that a part $\frac{p}{n}$ is completed. Without this assumption, the problem is inapproximable.

For the off-line study, we can simplify the model and have only two processor states, UP (also denoted by u) and $RECLAIMED$ (also denoted by r). Indeed, suppose that processor P_q is $DOWN$ for the first time at time-slot t: $\mathcal{S}_q[t] = d$. We can replace P_q by two 2-state processors $P_{q'}$ and $P_{q''}$ such that: 1) for all $j < t$, $\mathcal{S}_{q'}[j] = \mathcal{S}_q[j]$ and $\mathcal{S}_{q''}[j] = r$, 2) $\mathcal{S}_{q'}[t] = \mathcal{S}_{q''}[t] = r$, and 3) for all $j > t$, $\mathcal{S}_{q'}[j] = r$ and $\mathcal{S}_{q''}[j] = \mathcal{S}_q[j]$. In this way, we remove a $DOWN$ state and add a two-state processor. If we do this modification for each $DOWN$ state, we obtain an instance with only UP or $RECLAIMED$ processors. In the worst case, the total number of processors is multiplied by N, which does not affect the problem's complexity (polynomial versus NP-hard). Let OFF-LINE denote the problem of minimizing the time to complete the first iteration, with same-speed processors:

Theorem 4.1. *Problem* OFF-LINE *is NP-hard.*

Proof. Consider the associated decision problem: given a number m of tasks, of computing cost w and communication cost T_{data}, a program of communication cost T_{prog}, and a platform of p processors, with availability vectors $\mathcal{S}_q$, a bound n_{com} on the number of simultaneous communications, and a time limit N, does there exist a schedule that executes one iteration in time less than N? The problem is in NP: given a set of tasks, a platform, a time limit and a schedule (of communications and computations), we can check the schedule and compute its completion time with polynomial complexity.

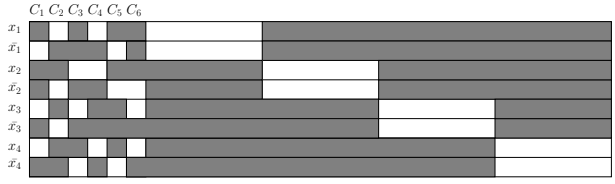

Figure 4.1: Instance $\mathcal{I}_2$ in proof of NP-completeness of OFF-LINE.

The (surprisingly difficult to establish) proof follows from a reduction from 3SAT. Let $\mathcal{I}_1$ be an instance of 3SAT : given a set $U = \{x_1, ..., x_n\}$ of variables and a collection $\{C_1, ..., C_m\}$ of clauses, does there exist a truth assignment of U? We suppose that each variable is present in at least one clause.

We construct the following instance $\mathcal{I}_2$ of the OFF-LINE problem with m tasks and $p = 2n$ processors: $n_{com} = 1$, $T_{\text{prog}} = m$, $T_{\text{data}} = 0$, $w_i = w = 1$, $N = m(n+1)$ and $\forall i \in [1,n], \forall j \in [1,m]$, 1) if $x_i \in C_j$ then $\mathcal{S}_{2i-1}[j] = u$ else $\mathcal{S}_{2i-1}[j] = r$, 2) if $\bar{x_i} \in C_j$ then $\mathcal{S}_{2i}[j] = u$ else $\mathcal{S}_{2i}[j] = r$, 3) $\mathcal{S}_{2i}[mi+j] = \mathcal{S}_{2i-1}[mi+j] = u$ and 4) $\forall k \in [1,n], i \neq k$, $\mathcal{S}_{2k-1}[mi+j] = \mathcal{S}_{2k}[mi+j] = r$. The size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$. Figure 4.1 illustrates this construction for $\mathcal{I}_1 = (\bar{x_1} \vee x_3 \vee x_4) \wedge (x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_2 \vee x_3 \vee \bar{x_4}) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x_1} \vee \bar{x_2} \vee \bar{x_4}) \wedge (\bar{x_2} \vee x_3 \vee x_4)$, with white meaning UP and grey meaning $RECLAIMED$.

Suppose that $\mathcal{I}_1$ has a solution A with, for all $j \in [1,n]$, $x_j = A[j]$. For any $i \in [1,m]$, there exists at least one true literal of A in C_i. We pick one arbitrarily. Let x_j be the associated variable. Then during time-slot i, if $A[j] = 1$, processor P_{2i-1} will download (a fraction of) the program, while if $A[j] = 0$, processor P_{2i} will download it. During this time-slot, no other processor communicates with the master. Then, for all $i \in [1,n]$:

- if $A[j] = 1$, between $mi+1$ and $m(i+1)$, P_{2i-1} completes the reception of the program and then executes as many tasks as possible, P_{2i} stays idle
- if $A[j] = 0$, then P_{2i-1} is idle and P_{2i} completes downloading its copy of the program and computes as many tasks as possible.

For all $i \in [1,2n]$, let L_i be the number of communication time-slots for processor P_i between time-slots 1 and m. By the choice of the processor receiving the program at any time-slot $t \in [1,m]$, if $A[i] = 0$, then $L_{2i-1} = 0$ and $L_{2i} = 1$, else $L_{2i} = 0$ and $L_{2i-1} = 1$. Let, for all $i \in [1,n]$, $p(i) = 2i - A[i]$. Then, for all $i \in [1,n]$, between time $mi+1$ and $m(i+1)$, $P_{p(i)}$ is available and $P_{p(i)+2A[i]-1}$ is idle. $P_{p(i)}$ completes receiving its program at the latest at time $mi + T_{prog} - L_{p(i)} = m(i+1) - L_{p(i)}$ and can execute $L_{p(i)}$ tasks before being reclaimed. Overall, the processors execute $X = \sum_{i=1}^{n} L_{p(i)}$ tasks. For any $j \in [1,m]$, by construction there is exactly one processor downloading the program during time-slot j. Consequently, $X = m$, and thus $\mathcal{I}_2$ has a solution.

Suppose now that $\mathcal{I}_2$ has a solution. As $n_{com} = 1$, for all $i \in [1,n]$, processors P_{2i-1} and P_{2i} receive the program during at most m time slots before time m. After time m, processors P_{2i-1} and P_{2i} are only available between time $mi+1$ and $m(i+1)$. Then, at time N, the sum S of the time-slots spent in receiving the program on processors P_{2i-1} and P_{2i} is $S \leq 2m$. This means that at most one of these processors can execute tasks before time N. Among P_{2i-1} and P_{2i}, let $p(i)$ be the processor computing at least one task if any, and if no one execute a task, any of these processors. If neither P_{2i-1} nor P_{2i} computes any task, let $p(i) = 2i$. Let A be an array of size n such that if $p(i) = 2i-1$, then $A[i] = 1$ else $A[i] = 0$. We will prove that all clauses of $\mathcal{I}_1$ are satisfied with this assignment. Without loss of generality we assume that no communication is made to a processor that does not execute any task. Suppose that for $i \in [1,m]$, a processor P_j with $j = p(k)$ receives a part of the program during time-slot i. Then, by definition of the function p, either $A[k] = 1$ and $x_k \in C_i$, or $A[k] = 0$ and $\bar{x_k} \in C_i$. This means that assignment A satisfies clause C_i. Let X be the number of true clauses with assignment A. For all $i \in [1,2n]$, we define L_i as the number of communication time-slots for processor P_i between times 1 and m. Then, $X \geq \sum_{j=1}^{n} L_{p(i)}$. In addition, processor $P_{p(i)}$ completes the reception of the program at the latest at time $m(i+1) - L_i$, and then computes at most L_i tasks before being reclaimed at time $m(i+1)$. Overall, the processors compute m tasks. Then, $\sum_{j=1}^{n} L_{p(i)} \geq m$, and $X \geq m$. Consequently, all clauses are satisfied by A, i.e., $\mathcal{I}_1$ has a solution, which concludes the proof. ■

Proposition 4.1. *Problem* OFF-LINE *cannot be approximated within* $\frac{8}{7} - \epsilon$ *for all* $\epsilon > 0$.

Proof. MAXIMUM 3-SATISFIABILITY cannot be approximated within $\frac{8}{7} - \epsilon$ for all $\epsilon > 0$ [42]. The result is immediate for problem OFF-LINE by construction of the proof of Theorem 4.1. ■

Now we show that the difficulty of problem OFF-LINE is due to the bound n_{com}: if we relax this

bound, the problem becomes polynomial.

Proposition 4.2. OFF-LINE *is polynomial when* $n_{com} = +\infty$*, even with different-speed processors.*

Proof. Consider a strategy that sends the program to processors as soon as possible, at the beginning of the execution. Then, task by task, it greedily assigns the next task to the processor that can terminate its execution the soonest; this is the classical MCT (Minimum Completion Time) strategy, whose complexity is $m \times p$.

To show that this strategy is optimal, let S_1 be an optimal schedule, and S_2 a MCT schedule. Let T_1 and T_2 be the associated completion times. We aim at proving that $T_2 = T_1$. We first modify the schedule S_1 as follows. Suppose that processor P_q begins a computation or a communication at time t, and that it is available but idle during time-slot $t-1$. Then, we can shift forward the operation and execute it at time $t-1$ without breaking any rules and without increasing the completion time of the current iteration. We repeat this modification as many times as possible, and finally obtain a schedule S_1' with completion time $T_1' = T_1$. Assume now that P_q executes i tasks under schedule S_1' and j under S_2. The first $\min\{i,j\}$ tasks are executed at the same time by S_1' and by S_2. Suppose that $T_2 > T_1$. Consider S_2 right before the allocation of the first task whose completion time is $t > T_1$. At this time, at least one processor P_{q_0} has strictly fewer tasks in S_2 than in S_1'. We can thus allocate a task to P_{q_0} with completion time $t \leq T_1$. The MCT schedule should have chosen the latter allocation, and we obtain a contradiction. The MCT schedule S_2 is thus optimal. ■

The MCT algorithm is not optimal if $n_{com} < +\infty$. Consider an instance with $T_{\text{prog}} = T_{\text{data}} = 2$, two tasks ($m = 2$) and two identical processors ($p = 2$, $w_q = w = 2$). Suppose that $n_{com} = 1$, and that $\mathcal{S}_1 = [u, u, u, u, u, u, r, r, r]$ and $\mathcal{S}_2 = [r, u, u, u, u, u, u, u, u]$. The optimal schedule computes both tasks in time 9 as follow: stay idle for one time-slot and then send the program and data to P_2. However, MCT executes the first task on P_1, and is thus not optimal.

4.5 Computing the expectation

In this section, we first introduce a Markov model for processor availability, and then show how to compute the *expected execution time* of a processor to complete a given workload and the probability of success of its computations. Both these quantities will guide the design of some on-line heuristics in Section 4.6.

4.5.1 Expected execution time

The availability of processor P_q is described by a 3-state recurrent aperiodic Markov chain, defined by 9 probabilities: $P_{i,j}^{(q)}$, with $i, j \in \{u, r, d\}$, is the probability for P_q to move from state i at time-slot t to state j at time-slot $t+1$, which does not depend on t. We denote by $\pi_u^{(q)}$, $\pi_r^{(q)}$ and $\pi_d^{(q)}$ the limit distribution of P_q's Markov chain (i.e., steady-state fractions of state occupancy for states UP, $RECLAIMED$, and $DOWN$). This limit distribution is easily computed from the transition probability matrix, and $\pi_u^{(q)} + \pi_r^{(q)} + \pi_d^{(q)} = 1$.

When designing heuristics to assign tasks to processors, it seems important to take into account the expected execution time of a processor until it completes all tasks assigned to it. Indeed, speed is not the only factor, as the target processor may well become $RECLAIMED$ several times before executing all its scheduled computations. We develop an analytical expression for such an expectation as follows.

Consider a processor P_q in the UP state at time t, which is assigned a workload that requires W time-slots in the UP state for completing all communications and/or computations. To complete the

workload, P_q must be UP during another $W-1$ time-slots. It can possibly become $RECLAIMED$ but never $DOWN$ in between. What is the probability of the workload being completed? And, if it is completed, what is the expectation of the number of time-slots until completion?

Definition 4.1. *Knowing that P_q is UP at time-slot t_1, let $\mathbf{P_+^{(q)}}$ be the conditional probability that it will be UP at a later time-slot, without going to the $DOWN$ state in between. Formally, knowing that $\mathcal{S}_q[t_1] = u$, $P_+^{(q)}$ is the conditional probability that there exists a time t_2 such that $\mathcal{S}_q[t_2] = u$ and $\mathcal{S}_q[t] \neq d$ for $t_1 < t < t_2$.*

Definition 4.2. *Let $\mathbf{E^{(q)}(W)}$ be the conditional expectation of the number of time-slots required by P_q to complete a workload of size W knowing that it is UP at the current time-slot t_1 and will not become $DOWN$ before completing this workload. Formally, knowing that $\mathcal{S}_q[t_1] = u$, and that there exist $W-1$ time-slots $t_2 < t_3 < \cdots < t_W$, with $t_1 < t_2$, $\mathcal{S}_q[t_i] = u$ for $i \in [2, W]$, and $\mathcal{S}_q[t] \neq d$ for $t \in [t_1, t_W]$, $E^{(q)}(W)$ is the conditional expectation of $t_W - t_1 + 1$.*

Lemma 4.1. $P_+^{(q)} = P_{u,u}^{(q)} + \frac{P_{u,r}^{(q)} P_{r,u}^{(q)}}{1-P_{r,r}^{(q)}}$.

Proof. The probability that P_q will be available again before crashing is the probability that it remains available during the next time-slot, plus the probability that it becomes $RECLAIMED$ and later returns to the UP state before crashing. We obtain that

$$P_+^{(q)} = P_{u,u}^{(q)} + P_{u,r}^{(q)} \left(\sum_{t=0}^{+\infty} (P_{r,r}^{(q)})^t \right) P_{r,u}^{(q)},$$

hence the result. ■

Theorem 4.2. $E^{(q)}(W) = W + (W-1) \times \frac{P_{u,r}^{(q)} P_{r,u}^{(q)}}{1-P_{r,r}^{(q)}} \times \frac{1}{P_{u,u}^{(q)}(1-P_{r,r}^{(q)}) + P_{u,r}^{(q)} P_{r,u}^{(q)}}$.

Proof. To execute the whole workload, P_q needs $W-1$ additional time-slots of availability. Consequently, the probability that P_q successfully executes its entire workload before crashing is $(P_+^{(q)})^{W-1}$.

The key idea to prove the result is to consider $E^{(q)}(up)$, the expected value of the number of time-slots before the next UP time-slot of P_q, knowing that it is up at time 0 and will not become $DOWN$ in between:

$$E^{(q)}(up) = \frac{P_{u,u}^{(q)} + \sum_{t \geq 0} (t+2) P_{u,r}^{(q)} (P_{r,r}^{(q)})^t P_{r,u}^{(q)}}{P_+^{(q)}}.$$

To compute $E^{(q)}(up)$, we study the value of

$$A = \sum_{t \geq 0} (t+2) P_{u,r}^{(q)} (P_{r,r}^{(q)})^t P_{r,u}^{(q)} = \frac{P_{u,r}^{(q)} P_{r,u}^{(q)}}{P_{r,r}^{(q)}} \sum_{t \geq 0} (t+2) (P_{r,r}^{(q)})^{t+1} = \frac{P_{u,r}^{(q)} P_{r,u}^{(q)}}{P_{r,r}^{(q)}} g'(P_{r,r}^{(q)})$$

with $g(x) = \sum_{t \geq 0} x^{t+2} = \frac{x^2}{1-x}$. Differentiating, we obtain $g'(x) = \frac{x(2-x)}{(1-x)^2}$ and

$$A = \frac{P_{u,r}^{(q)} P_{r,u}^{(q)}}{P_{r,r}^{(q)}} \times \frac{P_{r,r}^{(q)} (2 - P_{r,r}^{(q)})}{(1 - P_{r,r}^{(q)})^2}.$$

Letting $z = \frac{P_{u,r}^{(q)} P_{r,u}^{(q)}}{P_{u,u}^{(q)}(1-P_{r,r}^{(q)})}$, we derive

$$E^{(q)}(up) = \frac{1 + z\frac{(2-P_{r,r}^{(q)})}{(1-P_{r,r}^{(q)})}}{1+z} = 1 + \frac{z}{(1-P_{r,r}^{(q)})(1+z)}$$

We then conclude by noting that $E^{(q)}(W) = 1 + (W-1) \times E^{(q)}(up)$. ■

4.6 On-line heuristics

4.6.1 Rationale

In this section, we propose heuristics to address the on-line version of the problem. Conceptually, we can distinguish three main classes of heuristics:

Passive heuristics that conservatively keep current processors active as long as possible: the current configuration is changed only when one of the enrolled processors becomes $DOWN$.

Dynamic heuristics that may change configuration on the fly, while preserving ongoing work. More precisely, if a processor is engaged in a computation, it finishes it; if it is engaged in a communication, it finishes it together with the corresponding computation. But otherwise, tasks can be freely reassigned among processors, whether already enrolled or not. Intuitively, the idea is to benefit from, say, a fast and reliable resource that has just become UP, while not risking losing part of the work already completed for the current iteration.

Proactive heuristics that would allow for the possibility of aggressively terminating ongoing tasks, at the risk for an iteration to never complete.

In our model, the dynamic strategy is the most appealing. Since tasks are executed one by one and independently on each processor, using a passive approach by which all m tasks are assigned once and for all without possible reassignment does not make sense. A proactive strategy would have little impact on the time to complete the iteration unless the last tasks are assigned to slow processors. In this case, these tasks should be terminated and assigned to faster processors, which could have significant benefit when m is small. A simpler and popular solution is to use only dynamic strategies but to *replicate* these last tasks on one or more hosts in the UP state, canceling all remaining replicas when one of them completes. Concerning communications, the priority is given to one replica by task. Task replication may seem wasteful, but it is a commonly used technique in desktop grid environments in which resources are plentiful and often free of charge. While never detrimental to execution time, task replication is more beneficial when m is small.

In all the heuristics described hereafter, a task is replicated whenever there are more processors in the UP state than there are remaining tasks to execute. We limit the number of additional replicas of a task to two, which has been used in previous work [53] and works well in our experiments (better performance than with only one additional replica, not significantly worse performance than with more additional replicas). For simplicity, we describe all our heuristics assuming no task replication, but it is to be understood that there are up to $3m$ tasks (instead of m) distributed by the master during each iteration; the m original tasks are given priority over replicas, which are scheduled only when room permits.

All heuristics assign tasks to processors (that must be in the UP state) one-by-one, until m tasks are assigned. More precisely, at time slot t, there are enrolled processors that are currently active, either receiving some message, or computing a task, or both. Let m' be the number of tasks whose communication or computation has already begun at time t. Since ongoing activities are never terminated, there

remain $m-m'$ tasks to assign to processors. The objective of the heuristics is to decide which processors should be used for these tasks.

The dynamic heuristics below fall into two classes, *random* and *greedy*. Most of these heuristics rely on the assumption that processor availability follows a Markov process, as discussed in Section 4.5.

4.6.2 Random heuristics

The heuristics described in this section use randomness to select which processor, among the ones that are in the UP state, will execute the next task. The simplest heuristic, RANDOM, assigns the next task to a processor picked randomly using a uniform probability distribution. Going beyond RANDOM, it is possible to assign a weight to processor P_q, in a view to giving larger weight to more "reliable" processors. Processors are picked with a probability equal to their normalized weights. We propose four ways of defining these weights:

1. **Long time** UP: the weight of P_q is $P_{u,u}^{(q)}$, the probability that P_q remains UP, hence favoring processors that stay UP for a long time.
2. **Likely to work more**: the weight of P_q is $P_{+}^{(q)}$, the probability that P_q will be UP another time slot before crashing (see Section 4.5), hence favoring processors with high probability of becoming UP again before crashing.
3. **Often** UP: the weight of P_q is $\pi_u^{(q)}$, the steady-state fraction of time that P_q is UP, hence favoring processors that are UP more often.
4. **Rarely** $DOWN$: the weight of P_q is $(1-\pi_d^{(q)})$, hence favoring processors that are $DOWN$ less often.

We call the corresponding heuristics RANDOM1, RANDOM2, RANDOM3, and RANDOM4. For each of these four heuristics P_q's weight can be divided by w_q, attempting to account for processing speed as well as reliability. We thus obtain four additional variants, designed by the suffix 'W'.

4.6.3 Greedy heuristics

We propose four general heuristics, each of which can be enhanced to account for network contention.

MCT (Minimum Completion Time)

Assigning a task to the processor that can complete it the soonest is the optimal policy in the offline case without network contention (Proposition 4.2). We apply MCT here as follows. For each processor P_q we compute Delay(q), the delay before P_q finishes its current activities, and after which it could be enrolled for one of the $m-m'$ remaining tasks to be scheduled. In addition to processors finishing ongoing work, other processors could need to receive all or part of the program. Because of processors becoming $RECLAIMED$, we cannot exactly compute Delay(q). As a first approach, we estimate it assuming that P_q remains in the UP state and that there is no network contention whatsoever. We then greedily assign each of the remaining $m-m'$ tasks to processors, picking each time the processor with the smallest task completion time. More formally, for each processor P_q, let n_q be the number of tasks already assigned to it (out of the $m-m'$ tasks), and let $CT(P_q, n_q)$ be the estimation of its completion time:

$$CT(P_q, n_q) = \text{Delay}(q) + T_{\text{data}} + \max(n_q - 1, 0)\max(T_{\text{data}}, w_q) + w_q \ . \tag{4.1}$$

MCT assigns the next task to processor P_{q_0}, where $q_0 = \text{ArgMin}\{CT(P_q, n_q + 1)\}$.

MCT with contention – The estimated completion time in Equation 4.1 does not account for network contention (caused by the master's limited network capacity). Because of the overlap between communications and computations, it is difficult to predict network traffic. Instead, we use a simple correcting factor, and replace T_{data} by $\left\lceil \frac{n_{\text{active}}}{n_{com}} \right\rceil T_{\text{data}}$, where n_{active} denotes the number of active processors, i.e., those processors that have been assigned one or several of the $m - m'$ tasks. The n_{active} counter is initialized to zero and is incremented when a task is assigned to a newly enrolled processor. The intuition is that this counter measures the average slowdown encountered by a worker when communicating with the master. This estimation is simple but pessimistic since all scheduled communications do not necessarily take place simultaneously. We derive the new estimation:

$$CT(P_q, n_q) = \text{Delay}(q) + \left\lceil \frac{n_{\text{active}}}{n_{com}} \right\rceil T_{\text{data}} + \max(n_q - 1, 0) \max(\left\lceil \frac{n_{\text{active}}}{n_{com}} \right\rceil T_{\text{data}}, w_q) + w_q \qquad (4.2)$$

We call MCT* the version of the MCT heuristic that uses the above definition of $CT(P_q, n_q)$.
Expected MCT – Given a workload (i.e., a number of needed time-slots of computation) $CT(P_q, n_q)$, Theorem 4.2 gives the value of $E^{(q)}(CT(P_q, n_q))$, the expected number of time-slots needed for P_q to be UP during $CT(P_q, n_q)$ times-slots without becoming $DOWN$ in between. Using this expectation as the criterion for selecting processors, and depending on whether the correcting factor on T_{data} is used, we obtain one new version of MCT and one new version of MCT*, which we call EMCT and EMCT*, respectively.

LW (Likely to Work)

We build heuristics that consider the probability that a processor P_q, which is UP, will be UP again at least once before becoming $DOWN$. This probability, $P_+^{(q)}$, is given by Lemma 4.1. We assign the next task to processor P_{q_0} with the highest probability of being UP for at least the estimated number of needed time-slots to complete its workload, before becoming $DOWN$:

$$q_0 = \text{ArgMax} \left\{ (P_+^{(q)})^{CT(P_q, n_q+1)} \right\} .$$

Therefore, we first estimate the size $\mathcal{W}$ of the workload and then the probability that a processor will be in the UP state $\mathcal{W}$ time-slots without becoming $DOWN$ in between. Using Equation 4.2 instead of Equation 4.1, one obtains the LW* heuristic.

UD (Unlikely Down)

Here, we estimate the number $\mathcal{N}$ of time-slots needed for a processor to complete its workload, knowing that it can become $RECLAIMED$. Then we compute the probability that it will not become $DOWN$ for $\mathcal{N}$ time-slots. Given that P_q starts in the UP state, the probability that it does not go to the $DOWN$ state during k time-slots is:

$$P_{UD}^{(q)}(k) = \begin{bmatrix} 1 & 1 \end{bmatrix} . \begin{bmatrix} P_{u,u}^{(q)} & P_{u,r}^{(q)} \\ P_{r,u}^{(q)} & P_{r,r}^{(q)} \end{bmatrix}^{k-1} . \begin{bmatrix} 1 \\ 0 \end{bmatrix} .$$

We approximate this expression by forgetting the state of P_q after the first transition:

$$P_{UD}^{(q)}(k) = (1 - P_{u,d}^{(q)}) \left(1 - \frac{P_{u,d}^{(q)} \pi_u^{(q)} + P_{r,d}^{(q)} \pi_r^{(q)}}{\pi_u^{(q)} + \pi_r^{(q)}} \right)^{k-2} .$$

We use this value with $k = E^{(q)}(CT(P_q, n_q + 1))$. UD assigns the next task to the processor P_{q_0} that maximizes the probability of not becoming $DOWN$ before the estimated number of time-slots needed for it to complete its workload, counting the time-slots spent in the $RECLAIMED$ state:

$$q_0 = \text{ArgMax}\{P_{UD}^{(q)}(E^{(q)}(CT(P_q, n_q + 1)))\} .$$

Using Equation 4.2 instead of Equation 4.1, one obtains the UD* heuristic.

4.7 Experiments

We have evaluated the heuristics described in the previous section using a discrete-even simulator for the execution of application on volatile resources (The simulator is publicly available at `http://graal.ens-lyon.fr/~fdufosse/changing_platforms.tar.gz`). The simulator takes as input values for all the parameters listed in Section 4.3, and it assumes that temporal processor availability follows a Markov process.

For the simulation experiments, rather than fixing N, the number of time-slots, we instead fix the number of iterations to 10. The quality of an application execution is then measured by the time needed to complete 10 iterations, or *makespan*. This equivalent problem is simpler to instantiate since it does not require choosing meaningful N values, which would depend on the application and platform characteristics. We have executed all heuristics presented above for several problem instances. For each problem instance we compute the *degradation from best* (dfb) of each heuristic, i.e., the percentage relative difference between the makespan achieved by the heuristic and that achieved by the best heuristic, all for that particular instance. A value of zero means that the heuristic is best for the instance. We use this metric because makespans vary widely between instances depending on processor availability patterns. We also count how often, over all instances, each heuristic is the (or tied with the) best one, so that we can report on numbers of wins for each heuristics.

All our experiments are for $p = 20$ processors. The Markov chain that characterizes processor P_q's availability is defined as follows. We uniformly pick a random value between 0.90 and 0.99 for each $P_{x,x}^{(q)}$ value (for $x = u, r, d$). We then set $P_{x,y}^{(q)}$ to $0.5 \times (1 - P_{x,x}^{(q)})$, for $x \neq y$. An experimental scenario is defined by the above and by three parameters: n, the number of tasks per iteration, n_{com}, the constraint on the master's communication bandwidth, and the w_{min} parameter, which is used as follows. For each processor P_q, we pick w_q uniformly between w_{min} and $10 \times w_{min}$. T_{data} is set to w_{min}, meaning that the fastest possible processor has a computation-communication ratio of 1. T_{prog} is set to $5 \times w_{min}$, meaning that downloading the program takes 5 times as much time as downloading the data for a task. We define experimental scenarios for each of the possible instantiations of (n, n_{com}, w_{min}) given the values shown in Table 4.1. We must emphasize that our goal here is *not* to instantiate a representative model for a desktop grid and application, but rather to create arbitrary but simple synthetic experimental scenarios that will highlight inherent strengths and weaknesses of the heuristics.

For each possible instantiation of the parameters in Table 4.1, we create 247 random experimental scenarios as described above. For each experimental scenario, we run 10 trials, varying the seed of the random number generator used to determine Markov state transitions. We compute average dfb values for each heuristic based over these 10 trials, for each experimental scenarios. The total number of generated problem instances is $4 \times 3 \times 10 \times 247 \times 10 = 296,400$.

Table 4.2 shows average dfb and number of wins results, averaged over all experimental scenarios and sorted by increasing dfb values, i.e., from best to worst. In spite of the averaging over all problem instances, the trends are clear. All four MCT algorithms perform best, followed closely behind by the UD, and then the LW algorithms. The random algorithms perform significantly worse. Regarding these

Table 4.1: Parameter values for Markov experiments.

parameter	values
p	20
n	$5, 10, 20, 40$
n_{com}	$5, 10, 20$
w_{min}	$1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

Table 4.2: Results over all problem instances

Algorithm	Average dfb	#wins
EMCT	4.77	80320
EMCT*	4.81	78947
MCT	5.35	73946
MCT*	5.46	70952
UD*	7.06	42578
UD	8.09	31120
LW*	11.15	28802
LW	12.74	19529
RANDOM1W	28.42	259
RANDOM2W	28.43	301
RANDOM4W	28.51	278
RANDOM3W	31.49	188
RANDOM3	44.01	87
RANDOM4	47.33	88
RANDOM1	47.44	36
RANDOM2	47.53	73
RANDOM	47.87	45

algorithms, one can note that, expectedly, biasing the probability that a processor P_q is picked by w_q is a good idea (i.e., RANDOMxW always outperforms RANDOMx). The other differences in the definitions of the random algorithms do not lead to significant performance differences. On average on all problem instances, EMCT algorithms have makespans 10% smaller than the MCT algorithms, which shows that taking into account the probability of state changes does lead to improved performance.

To provide more insight than the overall averages shown in Table 4.2, Figure 4.2 plots dfb results averaged for distinct w_{min} values, shown on the x-axis. We present only results for the four MCT heuristics and for those heuristics that do account for network contention (i.e., with a $*$), and leave out the random heuristics. Note that increasing w_{min} amounts to scaling the unit time, meaning that availability state transitions occur more often during the execution of a task. In other words, the right hand side of the x-axis in Figure 4.2 corresponds to more difficult problem instances. Indeed, the larger w_{min}, the higher the probability that a task's processor experiences a state transition. Therefore, as w_{min} increases, it becomes increasingly important to estimate the negative impacts of the $DOWN$ and $RECLAIMED$ states: the most powerful processor may no longer be the best choice if it has a higher probability of going into the states $RECLAIMED$ or $DOWN$. The two EMCT algorithms take into account the probability that a processor enters the $RECLAIMED$ state. We see that they overtake the MCT algorithms when w_{min} becomes larger than 3. The UD and LW algorithms also take into

Table 4.3: Results for contention-prone experiments

Communication times $\times 5$

Algorithm	Average dfb
EMCT*	3.87
MCT*	4.10
UD*	5.23
EMCT	6.13
UD	6.42
MCT	7.70
LW*	8.76
LW	10.11

Communication times $\times 10$

Algorithm	Average dfb
UD*	2.76
UD	3.20
EMCT*	3.66
LW*	4.02
MCT*	4.22
LW	4.46
EMCT	8.02
MCT	15.50

account the probability that a processor goes $DOWN$. UD heuristics consistently outperform their LW counterparts. Also, UD (slightly) overtakes EMCT as soon as $w_{min} = 7$. We conclude that when the probability of state transitions rises one must use heuristics that explicitly take into account that processors can go in the states $RECLAIMED$ and $DOWN$.

In our results, we do not see much difference between the original versions of the heuristics and the versions that try to account for network contention, i.e., the heuristics that have a '$*$' in their names. Part of the reason may be that, as stated in Section 4.6.3, the correcting factor used to account for contention is a very coarse approximation. However, our experimental scenarios correspond to compute-intensive executions, meaning that processors typically spend much more time computing than communicating. We ran a set of experiments for $n = 20$, $n_{com} = 5$, and $w_{min} = 1$, but with $T_{\text{data}} = 5w_{min}$ and $T_{\text{prog}} = 25w_{min}$, i.e., with communication times 5 times larger than those in our base set of experimental scenarios. Results averaged over 100 such "contention-prone" experimental scenarios (each of which is ran for 10 trials) are shown in the left-hand side of Table 4.3. The right-hand side shows similar results for communication that are 10 times larger than those in our base set of scenarios. These results confirm that, as the scenario becomes more communication intensive, those algorithms that account for network contention outperform their counterparts.

4.8 Conclusion

In this chapter, we have studied the problem of scheduling master-worker iterative applications on volatile platforms in which hosts can experience failures or be temporarily reclaimed by their owners. A unique aspect of our work is that we model the fact that communication between the master and the workers is subject to a bandwidth constraint, e.g., due to the limited capacity of the master's network card. In this context we have made a theoretical contribution by characterizing the computational complexity of the off-line problem, which turns out to be NP-hard. Interestingly, without any bandwidth constraint, the problem becomes solvable in polynomial time. We have then proposed several on-line scheduling heuristics. By assuming a Markov model of processor availability, we were able to derive a closed-form formula for the expectation of the time needed by a volatile worker to complete a set of tasks. Some of our heuristics use this expectation for making scheduling decision (namely EMCT, EMCT*, UD, UD*). Some heuristics also use a contention-correcting factor as a way to account for the constraint on the master's bandwidth (namely EMCT*, LW*, UD*). The evaluation of our heuristics in simulation has led to the following conclusions:

– Our failure-aware heuristics deliver better performance than classical heuristics when the probability that a task is subject to processor state transitions becomes non negligible;

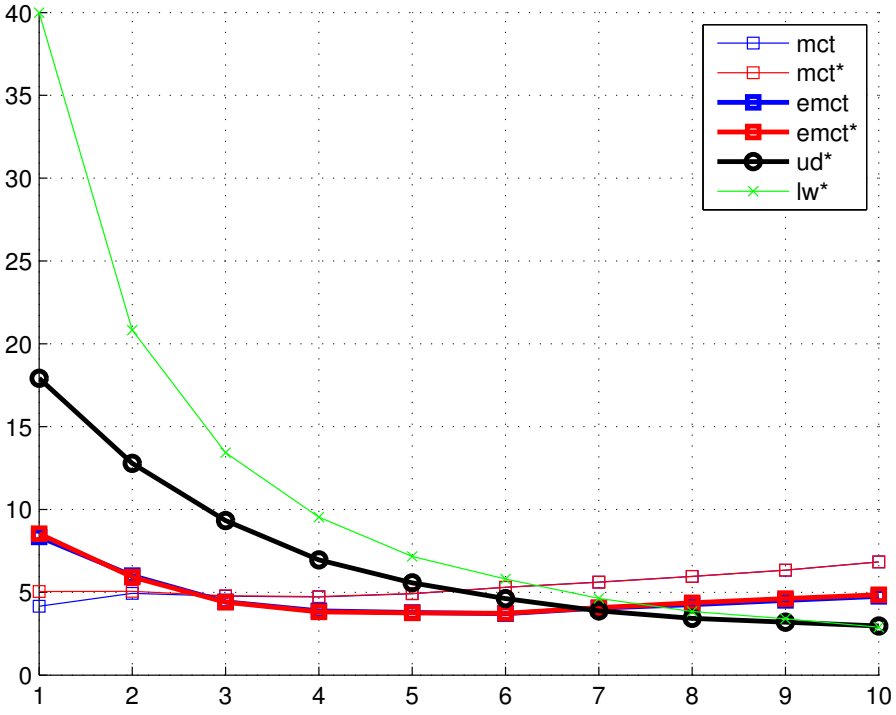

Figure 4.2: Averaged dfb results vs. w_{min}.

– Our contention-correcting factor improves performance on contention-prone platforms, and does not degrade performance otherwise;
– Our EMCT* heuristic delivers overall good performance, leading to a 10% reduction over the makespans achieved by MCT, the optimal algorithm for the contention-free offline case;
– EMCT* is outperformed by UD* in scenarios that exhibit very large state transition probabilities when compared to task duration, or a highly contented network.

The studied model only consider completely independent tasks. In many iterative applications, tasks have dependencies or need to communicate during its executions. Then, a new model has to be studied with more communications between tasks.

Chapter 5

Scheduling parallel iterative coupled applications on volatile resources

5.1 Introduction

This chapter deals with the problem of mapping *tightly-coupled* iterative applications onto volatile platforms. This work is a follow-on of the previous chapter, where we have addressed the deployment of iterative applications composed of *independent tasks* onto volatile platforms. The coupling of the application tasks induces new and challenging problems: one still needs to select resources that are either fast and/or reliable, but in addition these resources must be simultaneously available for the execution to make any progress.

As in the previous chapter, we envision a typical scientific iterative application. The master parallelizes the execution of the iterations across available processors. Each iteration corresponds to the execution of a fixed number of tasks, and there is a synchronization (or checkpoint) at the end of each iteration, before proceeding to the next one. However, while in the previous chapter all tasks were assumed to be independent, here we assume that all tasks are tightly coupled and keep exchanging information throughout the iteration. Note that these two generic models cover the two extreme points of the parallelization spectrum, and are representative of a very large class of scientific applications.

The platform model considered here is the same as in the previous chapter. We assume that the master can enroll a set of volatile computing workers. We use a model of availability with states UP, $DOWN$ and $RECLAIMED$. We bound the total outgoing communication capacity of the master, which is able to communicate simultaneously with only a limited number of workers, sending them either the application program or task data files. Given the application and platform models, and given a total execution time allotted to the application, the goal is to derive mapping strategies that maximize the expected number of iterations that will be successfully executed within the available time interval.

The major contribution of the chapter is twofold: on the theoretical side, we assess the complexity of the problem in its off-line version, i.e., when processor states are known in advance. Even with the global knowledge of resource availability, the problem is NP-hard: the communication bound forces combinatorial trade-offs between keeping former resources active, and enrolling new ones. On the practical side, we design a set of on-line heuristics which we thoroughly evaluate and compare using extensive simulations, based upon random parameters.

The chapter is organized as follows. Section 5.2 is devoted to a precise statement of the application and platform models. Complexity results for the off-line study are reported in Section 5.3. In Section 5.4, we describe our 3-state Markovian model of processor availability, and we show how to approximate the expected time for a processor to complete a given workload. Heuristics for the on-line problem

are detailed in Section 5.5. An experimental evaluation of these heuristics is conducted in Section 5.6. Finally, we give some conclusions and perspectives in Section 5.7.

5.2 Framework

In this section, we discuss the application and platform models in full details.

5.2.1 Application

As in the previous chapter, we target a scientific application that performs successive iterations. Each iteration corresponds to the execution of a fixed number m of tasks. There is a synchronization (or checkpoint) at the end of each iteration, before proceeding to the next one. Before being able to perform any computation, a processor needs:

– the program, of size V_{prog}, which is the same for all tasks and iterations
– the input data for each task, of constant size V_{data}, but that depends upon each task and iteration.

Unlike the previous case, the m same-size tasks steadily communicate throughout the iteration. All the m tasks of an iteration are executed concurrently, within the same time interval, and at the pace of the slowest resource. At any time slot of computation, execution of all tasks progress and the same ratio of each task is executed. If one processor computing some task fails, all the work executed for current iteration is lost, and computations of tasks have to be restarted. If one processor of current configuration is preempted, the computation of all tasks is interrupted. A given processor may well execute several tasks. There is no possible overlap between data reception and current computations. We can consider that an iteration consists in two step: a step of communication, and a step of computations.

We are given a number N of time-slots, and the objective is to compute as many iterations as possible within these time-slots. To simplify things, each task is executed within an integer number of time-slots. The length of a time-slot depends on the application/platform pair (see below), and can vary from a few minutes to a couple of days.

5.2.2 Configuration

Let $config(t)$ denote the set of enrolled processors, or *configuration*, at time t. If one of the processors crashes, all partial work done so far for the current iteration is lost, and the scheduler must select another set of processors to resume the iteration from scratch. But even if all processors in $config(t)$ are UP, the scheduler may well decide to change the configuration, for instance because a desirable (i.e., fast and/or reliable) processor just returned to the UP state. Changing configuration is a risky decision, because all partial work done so far is lost when a configuration change occurs, be it due to a processor crash or to a new resource selection. This is because all m tasks are executed concurrently in this model, so they all start, and eventually terminate, at the same time.

Similarly, the impact of a resource being reclaimed is important, because the whole configuration stalls until the resource eventually becomes UP again.

To summarize notations so far, we have:

– the time line is discretized into N time-slots
– m tasks are executed to complete each iteration of the application
– p processors may be enrolled; any processor P_q needs w_q time-slots to execute a task and its state vector $\mathcal{S}_q$ gives its state at each time-slot
– T_{prog} is the number of time-slots needed to send the program (of size V_{prog}) to a processor
– T_{data} is the number of time-slots needed to send the data of a task (of size V_{data}) to a processor

- n_{com} is the maximum number of processors that the master can communicate with (sending either program or input data) at each time-slot
- $config(t)$ is the configuration (i.e., the set of enrolled processors) at time-slot t

The objective is to assign tasks to processors so as to achieve the largest possible number of iterations within the N time-slots. Note that for the off-line study, if an efficient algorithm can be found to solve the previous problem, then, using binary search, an efficient algorithm can be designed to solve the problem of executing a given number of iterations in the minimum amount of time.

5.2.3 Execution scenario

To complete an iteration, enrolled processors must progress concurrently throughout the computations. But one resource may be assigned several tasks and execute them in parallel, if it has enough memory to do so. Formally, each processor P_q has a bound μ_q on the maximum number of tasks that it can execute simultaneously. We assume that $\sum_{q=1}^{p} \mu_q \geq m$, otherwise the problem has no solution. We derive the following conditions:

- The m tasks are mapped onto $k \leq m$ processors P_q. Each enrolled P_q is assigned x_q tasks, where $\sum_{q=1}^{k} x_q = m$. Because the tasks must proceed in locked steps, the execution progresses at the pace of the slowest resource. Hence the computation of an iteration requires $\max_q(x_q w_q)$ time-slots of concurrent computations (not necessarily consecutive, because of reclaiming).
- To be able to compute their tasks, the k enrolled processors must have received the program and all necessary data. More precisely:
  - Each resource P_q must receive the program, unless it has received it at some previous time-slot, and has not be $DOWN$ since that time-slot
  - In addition, each resource P_q must receive x_q data messages (one per task) from the master. This requires $x_q T_{\text{data}}$ time-slots if the master is available, and more if the master serves other processors (recall that the master cannot be simultaneously involved in more than n_{com} concurrent communications, be they program or data messages).

Note that the computation can start at a time-slot t only if each of the k enrolled resources has the program and the data of all its allocated tasks, and has remained UP or $RECLAIMED$ since receiving these messages. All processors must remain simultaneously UP from time-slot t up to time-slot $t + \max_q(x_q w_q) + t' - 1$ for some t': there must be $\max_q(x_q w_q)$ time-slots with all enrolled processors being simultaneously UP, and there may be t' time-slots with one or more processors $RECLAIMED$ in between (and the other UP). If this is the case, the iteration is validated, and the scheduler can start enrolling processors and sending tasks for the next iteration. Only those processors that have received the program and stayed UP or $RECLAIMED$ since that reception, need not to receive it again.

Changing configuration The scheduler may choose a new configuration at each time-slot t. Let P_q be a new enrolled resource at that point, i.e., $P_q \in config(t+1) \setminus config(t)$. P_q needs to receive the program unless it already has a copy of it and has not been $DOWN$ since its reception. In all cases, P_q need to receive the task data before starting to compute: this accounts for x_q messages. This holds true even if P_q had been enrolled at some previous time-slot $t' < t$, but also had been un-enrolled in between. In other words, an interrupted communication is resumed from scratch if the processor fails down, or if it was receiving task data, and it has been removed of the configuration.

5.2.4 Example

Consider an instance with $m = 5$ tasks and $p = 5$ heterogeneous processors with $\forall 1 \leq i \leq 5, w_i = i$. We suppose that $n_{com} = 2$, $T_{prog} = 2$ and $T_{data} = 1$. The state of processors is given in Figure 5.1 where colors have the following meaning: white means UP, grey means $RECLAIMED$ and black means $DOWN$.

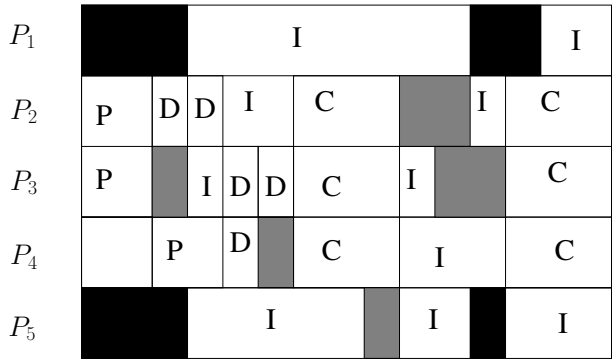

Figure 5.1: Example of execution of an iteration.

In the executed schedule, two tasks are assigned to processors P_2 and P_3, and the last one to P_4. This means a work load of 4 time slots on P_2, 6 time slots on P_3 and 4 time slots on P_4. Then, the computation in this configuration need 6 time slots of computations, and during each time slot of computation, $1/6$ of each task is executed.

The configuration is selected at time slot 1. At this time, P_1 and P_5 are not UP, so they cannot be selected in the configuration.

The communication step of this iteration is executed between time slot 1 and time slot 6. At time slot 1, the 3 processors selected can receive data, but because of bandwidth constraints, P_3 remains idle. In Figure 5.1, P means receiving the program, D means receiving the data of some task, and I means idle.

The computation step of the iteration is executed between time slot 7 and 14. At time slot 9, P_2 is preempted, and the computation is interrupted with half of the computation of each task completed. When P_2 becomes available again, P_3 has been preempted and the computation cannot restart. When P_3 becomes available again, P_2 and P_4 are UP, and the computation continues. If one of this processor had fail at time slot 14, all the computation would have been lost and a new step of communication would have starts. After time slot 14, a synchronization between processors is executed, and a new iteration can start at time slot 15.

5.3 Off-line complexity

In this section, we study the off-line complexity of the problem. This means that we assume the a priori knowledge of all processor states throughout the execution. In other words, the values $\mathcal{S}_q[j]$ are all known in advance, for $1 \leq q \leq p$ and $1 \leq j \leq N$.

For the off-line study, we can simplify the model and have only two processor states, UP (also denoted by 1) and $RECLAIMED$ (also denoted by 0). Indeed, suppose that processor P_q is $DOWN$ for the first time at time-slot t: $\mathcal{S}_q[t] = -1$. We can replace P_q by two 2-state processors $P_{q'}$ and $P_{q''}$ such that:

- for all $j \leq t$, $\mathcal{S}_{q'}[j] = \mathcal{S}_q[j]$ and $\mathcal{S}_{q''}[j] = 0$
- for all $j \geq t$, $\mathcal{S}_{q'}[j] = 0$ and $\mathcal{S}_{q''}[j] = \mathcal{S}_q[i]$

In this way, we remove a value -1 and add a two-state processor. If we do this modification for each value -1, we obtain an instance with only UP or $RECLAIMED$ processors. In the worst case, the total number of processors is multiplied by N, which does not affect the problem complexity (polynomial versus NP-hard).

In this section we show that the simplest off-line and deterministic versions of the problem are NP-hard. More precisely, the following two instances are both NP-hard:

- with $T_{\text{prog}} = T_{\text{data}} = 0$, $\mu_q = \mu = 1$, and $w_q = w$ (no communications, identical processors capable of executing a single task)
- with $T_{\text{prog}} = T_{\text{data}} = 0$, $\mu_q = +\infty$, and $w_q = w$ (no communications, identical processors capable of executing an arbitrary number of tasks)

Unlike Chapter 4, it has no sense to consider incomplete iterations for approximations. While the computation of tasks is not completed, all work currently done can be lost at next time slot. As minimizing the time to complete the first iteration with same-speed processors is NP-complete, the problem is inapproximable unless P=NP.

5.3.1 Fixed resource number

Without communications, if we use identical processors with $w_q = w$ and $\mu_q = \mu = 1$, we have to enroll m processors to complete an iteration. Then the problem reduces to finding w time-steps such that there exist m processors that are simultaneously UP during all these w time-steps. We call this problem OFF-LINE-COUPLED ($\mu = 1$).

Theorem 5.1. *Problem* OFF-LINE-COUPLED *($\mu = 1$) is NP-hard.*

Proof. The decision problem associated to OFF-LINE-COUPLED ($\mu = 1$) writes: given a value w and p state vectors $\mathcal{S}_q$, can we find m processors that are simultaneously UP during at least w time-steps? This problem clearly belongs to NP: the $m \times w$ sub-matrix is a certificate of polynomial (and even linear) size.

For the completeness, we use a reduction from ENCD, the Exact Node Cardinality Decision problem [22]. Let $\mathcal{I}_1$ be an instance of ENCD: given a bipartite graph $G = (V \cup W, E)$ and two integers a and b such that $1 \leq a \leq |V|$ and $1 \leq b \leq |W|$, does there exist a bi-clique with exactly a nodes in V and b nodes in W? Recall that a bi-clique $C = U_1 \cup U_2$ is a complete induced sub-graph: $U_1 \subset V$, $U_2 \subset W$, and for every $u_1 \in U_1, u_2 \in U_2$, the edge $(u_1, u_2) \in E$.

We construct the following instance $\mathcal{I}_2$ of OFF-LINE-COUPLED ($\mu = 1$): we let $p = |V|$ and $N = |W|$. Resource R_i (which corresponds to vertex $v_i \in V$) is UP at time-step j (which corresponds to vertex $w_j \in W$) if and only if $(v_i, w_j) \in E$. Finally we let $m = a$ and $w = b$. The size of the instance $\mathcal{I}_2$ is linear in the size of the instance $\mathcal{I}_1$. We show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does. Suppose first that $\mathcal{I}_1$ has a solution $C = U_1 \cup U_2$. We select the corresponding U_1 processors and the same U_2 time-steps. Because we have a clique, each resource is UP at each time-step, hence $\mathcal{I}_2$ has a solution. Suppose now that $\mathcal{I}_2$ has a solution. The corresponding sub-matrix translates into a bi-clique with a nodes in V and b nodes in W, hence a solution to $\mathcal{I}_2$. ■

5.3.2 Flexible resource number

Without communications, if we use identical processors with $w_q = w$ and $\mu_q = \mu = +\infty$ (in fact $\mu = m$ is enough), the task assignment problem is less constrained. Either we find m processors that are

simultaneously UP during w time-steps, or we find $\lceil \frac{m}{2} \rceil$ processors that are simultaneously UP during $2w$ time-steps, or again we find $\lceil \frac{m}{3} \rceil$ processors that are simultaneously UP during $3w$ time-steps, and so on. We call this problem OFF-LINE-COUPLED ($\mu = +\infty$).

Theorem 5.2. *Problem* OFF-LINE-COUPLED *($\mu = \infty$) is NP-hard.*

Proof. We use the same instance $\mathcal{I}_1$ of ENCD as in the proof of Theorem 5.1. We construct the following instance $\mathcal{I}_2$ of OFF-LINE-COUPLED ($\mu = +\infty$): we let $p = |V|$ and $N = 2|W| + 1$. Resource R_i (which corresponds to vertex $v_i \in V$) is UP at time-step $j \leq N$ (which corresponds to vertex $w_j \in W$) if and only if $(v_i, w_j) \in E$. All processors are up at each step j such that $|W| + 1 \leq j \leq N$. Finally we let $m = a$ and $w = b + |W| + 1$. Intuitively, this amounts to add $|W| + 1$ new vertices in W which are interconnected to every vertex in V. The size of the instance $\mathcal{I}_2$ is linear in the size of the instance $\mathcal{I}_1$. We show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ does. Suppose first that $\mathcal{I}_1$ has a solution $C = U_1 \cup U_2$. We select the corresponding U_1 processors and the same U_2 time-steps, plus the last $|W| + 1$ time-steps. We have $w = b + |W| + 1$, hence $\mathcal{I}_2$ has a solution. Suppose now that $\mathcal{I}_2$ has a solution. The corresponding sub-matrix translates into a bi-clique with x processors and y time-steps. If $x < m$ then at least one processor executes two tasks per iteration, and we need $2w$ times-steps to perform an iteration. But $2w > N$, what is a contradiction. Hence $x = m$ and $y = K$. At most $|W| + 1$ of the UP time-steps are greater than $|W|$, hence at least b of them are smaller than or equal to $|W|$: this leads to a solution to $\mathcal{I}_2$. ■

5.3.3 Polynomial instances

Remark. *Consider a problem instance without preemption, i.e. with processors either UP or $DOWN$. Some instances are polynomial, which do not contradict the above results because they are particular when we transform them into UP-$RECLAIMED$ instances as explained in the beginning of the previous chapter.*

The first particular case is without programs: $T_{prog} = 0$. Then the problem is polynomial: (i) we complete iterations as soon as possible; (ii) for any possible beginning and completion times of an iteration, we check if the processors which are UP during the whole can complete the iteration (complexity: $O(N^2)$ or $O(N \times W_{max})$).

The second polynomial case is with unbounded communications: $n_{com} = +\infty$. Any processor receives the program as soon as possible at the beginning and after failing down. Just as before, (i) we complete iterations as soon as possible; (ii) for any possible beginning and completion times of an iteration, we check if the processors which are UP during the whole time interval can complete the iteration; now this includes the time to receive data (complexity: $O(N^2)$ or $O(N \times W_{max})$).

Lemma 5.1. *The problem without preemption with $n_{com} = 1$ and $m = 1$ in the homogeneous case is polynomial.*

Proof. To solve this particular case, we use a dynamic programming algorithm that computes recursively the value $M(t, T_{comm}, P_i)$. It corresponds to the maximal possible number of iterations executed in the t first time slots with the last tasks executed on P_i with on the last processors T_{comm} time slots available for receiving the program. More precisely, suppose that processor P_i has successively executed many iteration, since a first reception of a first reception of a task data started at time slot t'. Between time slots t' and t, P_i has received task data, executed tasks and potentially has been idle some time slots. Then, T_{comm} is the number of time slots between t' and t during which P_i was computing some task, or was idle. These can be used by another processor to receive the program, while the remaining time, P_i was receiving task data, and no other communication with the master was possible.

We denote $T_f^t(i)$ the first time slot UP of P_i before t, that means the minimum value t' such that at any time slot between t' and t, P_i is in state UP ($\forall T_f^t(i) \leq t' \leq t, \mathcal{S}_i[t'] = 1$). We have to consider many cases to compute $M(t, T_{comm}, P_i)$.

In the fist case, the first iteration is executed on P_i (or no previous iteration interacts with these ones) and the last iteration is completed at time slot t. Then, P_i executes the n-th iteration at time $T_f^t(i) + T_{\text{prog}} + n \times (T_{\text{data}} + w)$ and with $T_{comm} = n \times w$. Then, $T_f^t(i) + T_{\text{prog}} + n \times (T_{\text{data}} + w) \leq t$ and $n \leq \lfloor \frac{t - T_f^t(i) - T_{\text{prog}}}{T_{\text{data}} + w} \rfloor$. Finally, $M(t, T_{comm}, P_i) = \lfloor \frac{t - T_f^t(i) - T_{\text{prog}}}{T_{\text{data}} + w} \rfloor$ if $T_{comm} \leq \lfloor \frac{t - T_f^t(i) - T_{\text{prog}}}{T_{\text{data}} + w} \rfloor \times w$, and 0 otherwise.

In the second case, the last iteration executed without P_i was computed by P_j and the reception of the program by P_i was completely overlapped by the computations on P_j. Let $t' < t$ be the time slot of completion of the last iteration on P_j. Then, if n is the number of iteration executed on P_i, $n \times (T_{\text{data}} + w) \geq t - t'$. Then, $M(t, T_{comm}, P_i) = M(t', T_{\text{prog}}, P_j) + \lfloor \frac{t - t'}{T_{\text{data}} + w} \rfloor$ if $T_{comm} \leq \lfloor \frac{t - t'}{T_{\text{data}} + w} \rfloor \times w$. In addition, P_i was available early enough to receive entirely the program before t', while P_j was executing iterations. Then, in this interval, P_j has executed T_{prog} time slots of computations, and $(\lceil \frac{T_{\text{prog}}}{w} \rceil - 1) \times T_{\text{data}}$ time slots of communications. Then $t' - T_f^{t'}(i) \geq T_{\text{prog}} + (\lceil \frac{T_{\text{prog}}}{w} \rceil - 1) \times T_{\text{data}}$

In the last case, the last iteration executed without P_i was computed by P_j and P_i had not completed the reception of the program at the end of this computation. Let t' be the time slot of the last iteration on P_j. Let T' be the number of time slots of reception of the program that P_i was able to do before t'. Then, $M(t, T_{comm}, P_i) = M(t', T', P_j) + \lfloor \frac{t - t' - (T_{\text{prog}} - T')}{T_{\text{data}} + w} \rfloor$ if $T_{comm} \leq \lfloor \frac{t - t' - (T_{\text{prog}} - T')}{T_{\text{data}} + w} \rfloor \times w$. As in the previous case, P_i was available early enough to receive T' time slots of the program, then $t' - T_f^{t'}(i) \geq T' + (\lceil \frac{T'}{w} \rceil - 1) \times T_{\text{data}}$

Then, in this three cases, we obtain this same formula:

$$\max_{\substack{P_j \neq P_i \\ t' \geq T_f^t(i) \\ 0 \leq T' \leq T_{\text{prog}}}} \left\{ M(t', T', P_j) + \left\lfloor \frac{t - t' - (T_{\text{prog}} - T')}{T_{\text{data}} + w} \right\rfloor \right\} \begin{array}{l} \text{if } T_{comm} \leq \lfloor \frac{t - t' - (T_{\text{prog}} - T')}{T_{\text{data}} + w} \rfloor \times w \\ \text{if } t' - T_f^t(i) \geq T' + (\lceil \frac{T'}{w} \rceil - 1) \times T_{\text{data}} \end{array}$$

The first case correspond to $T' = 0$, the second one to $T' = T_{\text{prog}}$, and the last one to $0 < T' < T_{\text{prog}}$.

Then, we obtain the following formula:

If $t = 0$, then $M(t, T_{comm}, P_i) = 0$.

If $\mathcal{S}_i[t] = 0$, then $M(t, T_{comm}, P_i) = M(t - 1, T_{comm} - 1, P_i)$. Otherwise:

$$M(t, T_{comm}, P_i) = \max \left\{ \begin{array}{l} \max_{\substack{P_j \neq P_i \\ t' \geq T_f^t(i) \\ 0 \leq T' \leq T_{\text{prog}}}} \left\{ M(t', T', P_j) + \left\lfloor \frac{t - t' - (T_{\text{prog}} - T')}{T_{\text{data}} + w} \right\rfloor \right\} \begin{array}{l} \text{if } T_{comm} \leq \lfloor \frac{t - t' - (T_{\text{prog}} - T')}{T_{\text{data}} + w} \rfloor \times w \\ \text{if } t' - T_f^t(i) \geq T' + (\lceil \frac{T'}{w} \rceil - 1) \times T_{\text{data}} \end{array} \\ M(t - 1, T_{comm} - 1, P_i) \end{array} \right\}$$

We do not consider the possibility that P_i receives the program during the computation on two different processors (for example on P_j and then on P_k). If P_i was able to receive some data before any reception by P_k, then P_i would have been a better choice than P_k to execute tasks after P_j, and if P_j has completed the communication during communications on P_k, it could have start an iteration at soon as it complete its reception.

■

Lemma 5.2. *The particular case without preemption with $T_{prog} = 0$ is polynomial.*

Proof. As $T_{prog} = 0$, the iterations are independent: completing each iteration as fast as possible is optimal. Iteration by iteration, we compute the values $F(T_{begin}, T_{end})$, that are equal to 1 if the iteration can be executed with beginning the computations at time slot T_{begin} and completing the computations at time slot T_{end}. In the other case, $F(T_{begin}, T_{end}) = 0$.

The algorithm to decide the value of $F(T_{begin}, T_{end})$, is the following:

Algorithm 13: Computation of $F(T_{begin}, T_{end})$

Data: an instance, a time slot T of beginning of the iteration and T_{begin} and T_{end}
Result: $F(T_{begin}, T_{end})$
Let S be the set of processors available between time slots T_{begin} and T_{end};
For all $P_u \in S$, P_u became available at $T_u(1) \leq T_{begin}$ and fail at $T_u(2) \geq T_{end}$;
We order the processors of S in decreasing order of $T_u(1)$;
foreach *processor* P_u **do**
  Let $n_t = \left\lfloor \frac{T_{end} - T_{begin}}{w_u} \right\rfloor$;
  n_t is the maximal number of tasks that P_u can execute between T_{begin} and T_{end};
  if *It remains some available time slots between* T *and* T_{begin} *to receive data* **then**
    Give the maximal number n_d data as possible with respect with $n_d \leq n_t$;
    Receive the task data as late as possible;
  end
  if *All tasks are executed* **then**
    Return 1;
  end
end
Return 0;

It remains to compute for all possible values of T_{begin} and T_{end}, the minimal T_{end} such that it exists a value T_{begin} with $F(T_{begin}, T_{end}) = 1$.

■

5.4 Computing the expectation of a workload

In this section, we show how to compute the expected execution time of a processor to complete a given workload on a given configuration.

As in Chapter 4, the availability of processor P_q is described by a 3-state recurrent aperiodic Markov chain, defined by 9 probabilities: $P_{i,j}^{(q)}$, with $i, j \in \{u, r, d\}$, is the probability for P_q to move from state i at time-slot t to state j at time-slot $t+1$, which does not depend on t. We denote by $\pi_u^{(q)}$, $\pi_r^{(q)}$ and $\pi_d^{(q)}$ the limit distribution of P_q's Markov chain (i.e., steady-state fractions of state occupancy for states UP, $RECLAIMED$, and $DOWN$). This limit distribution is easily computed from the transition probability matrix, and $\pi_u^{(q)} + \pi_r^{(q)} + \pi_d^{(q)} = 1$.

When designing heuristics to assign tasks to processor sets, it seems important to take into account the expected execution time of that processor set until it completes all tasks assigned to it or its probability of success. Indeed, speed is not the only factor, as the target processors need to be UP simultaneously to make any progress. Some of them may well become $RECLAIMED$ several times while others are UP, thereby delaying the completion of all their scheduled computations. We develop an analytical expression for such an expectation as follows.

5.4.1 Probability of success and expected cost of a computation

Consider a set S of processors, all in the UP state at time 0. This set is assigned a workload that requires W time-slots in the UP state for completing all computations. To complete the workload, all the processors in S must be simultaneously UP during another $W-1$ time-slots. They can possibly become $RECLAIMED$ (thereby freezing the execution) but never $DOWN$ in between. What is the probability of the workload being completed? And, if it is successfully completed, what is the expectation of the number of time-slots until completion?

Definition 5.1. *Knowing that all processors in a set S are UP at time-slot t_1, let $\mathbf{P_+^{(S)}}$ be the probability that they all will be UP simultaneously at a later time-slot, without any of them going to the $DOWN$ state in between. Formally, knowing that $\forall P_q \in S$, $\mathcal{S}_q[t_1] = u$, $P_+^{(S)}$ is the probability that there exists a time $t_2 > t_1$ such that*

$$\forall P_q \in S,\ \mathcal{S}_q[t_2] = u \text{ and } \mathcal{S}_q[t] \neq d \text{ for } t_1 < t < t_2$$

Definition 5.2. *Let $\mathbf{E^{(S)}(W)}$ be the conditional expectation of the number of time-slots required by a set of processors S to complete a workload of size W knowing that all processors in S are UP at the current time-slot t_1 and none will become $DOWN$ before completing this workload. Formally, knowing that $\mathcal{S}_q[t_1] = u$, and that there exist $W-1$ time-slots $t_2 < t_3 < \cdots < t_W$, with $t_1 < t_2$, $\mathcal{S}_q[t_i] = u$ for $i \in [2, W]$, and $\mathcal{S}_q[t] \neq d$ for $t \in [t_1, t_W]$, $E^{(S)}(W)$ is the conditional expectation of $t_W - t_1 + 1$.*

Theorem 5.3. *It is possible to numerically approximate the values of $P_+^{(S)}$ and $E^{(S)}(W)$ up to an arbitrary precision ε in fully polynomial time.*

Proof. Consider a set S of processors, all available at time slot 0. Consider the probability $P_+^{(S)}(t)$ that all these processors are simultaneously UP again for the first time at time t. This means that for all $0 < t' < t$, there exists at least one processor $RECLAIMED$ at time t'. Also, none of the processors in S goes $DOWN$ between 0 and t.

Let $P_{u \xrightarrow{t} u}^{(q)}$ be the probability that a processor P_q that was UP at time 0 is UP again at time t, without having been $DOWN$ in between, and let $P_{u \xrightarrow{t} u}^{(S)} = \prod_{P_q \in S} P_{u \xrightarrow{t} u}^{(q)}$. For each processor P_q, the value $P_{u \xrightarrow{t} u}^{(q)}$ can be computed by considering its transition matrix raised to the power t, knowing that the initial state is UP. We form the product to compute $P_{u \xrightarrow{t} u}^{(S)}$. We derive that

$$P_+^{(S)}(t) = P_{u \xrightarrow{t} u}^{(S)} - \sum_{0<t'<t} P_+^{(S)}(t') \times P_{u \xrightarrow{t-t'} u}^{(S)}.$$

The probability $P_+^{(S)}$ that all the processors in S will be simultaneously UP again at some point, before the first failure of any of them, is

$$\begin{aligned} P_+^{(S)} &= \textstyle\sum_{t>0} P_+^{(S)}(t) \\ &= \textstyle\sum_{t>0} P_{u \xrightarrow{t} u}^{(S)} - \sum_{0<t'<t} P_+^{(S)}(t') \times P_{u \xrightarrow{t-t'} u}^{(S)} \\ &= \textstyle\sum_{t>0} P_{u \xrightarrow{t} u}^{(S)} - \sum_{t>0} P_+^{(S)}(t) \times \sum_{t'>0} P_{u \xrightarrow{t'} u}^{(S)} \end{aligned}$$

Let $E_u(S) = \sum_{t>0} P_{u \xrightarrow{t} u}^{(S)}$. Suppose that all processors are UP at time slot 0. Let A_t the random variable that is equal to 1 if all processors are UP at time slot t without that ant processor goes $DOWN$ in between. Then $E(A_t) = P_{u \xrightarrow{t} u}^{(S)}$. By linearity of the expectation, we have $E(\sum_{0 \leq t' \leq t} A_{t'}) =$

$\sum_{0\leq t'\leq t} P^{(S)}_{\substack{t'\\ u\to u}}$. Suppose that, in set S, at least one processor has a nonzero probability of going $DOWN$. Then, $\lim_{t\to\infty}\sum_{0\leq t'\leq t} P^{(S)}_{\substack{t'\\ u\to u}}$ converges. We can conclude that $E(\sum_{t>0} A_t) = \sum_{t>0} P^{(S)}_{\substack{t\\ u\to u}}$. Then, $E_u(S)$ is the expected number of time slots with all processors UP, before one of these processors fails. Then, $P^{(S)}_+ = E_u(S) - E_u(S)\times P^{(S)}_+$, from which we derive that $P^{(S)}_+ = \frac{E_u(S)}{1+E_u(S)}$ if , in set S, at least one processor has a nonzero probability of going $DOWN$. Otherwise, $P^{(S)}_+ = 1$.

We now consider the expected time $E^{(S)}(W)$ to execute W time slots of computation, conditioned by the fact that no processor in S will fail. The first time slot of computation is done at $t=0$. Let $E^{(S)}_c$ be the expected time of the next time slot of computation. Then,

$$\begin{aligned} E^{(S)}_c &= \sum_{t>0} t\times P^{(S)}_+(t) \\ &= \sum_{t>0} t\times P^{(S)}_{\substack{t\\ u\to u}} - t\times\left(\sum_{0<t'<t} P^{(S}_+(t')\times P^{(S)}_{\substack{t-t'\\ u\to u}}\right) \\ &= \sum_{t>0} t\times P^{(S)}_{\substack{t\\ u\to u}} - \left(\sum_{t>0} P^{(S)}_+(t)\right)\times\left(\sum_{t'>0}(t+t')P^{(S)}_{\substack{t'\\ u\to u}}\right) \end{aligned}$$

Let $A(S) = \sum_{t>0} t\times P^{(S)}_{\substack{t\\ u\to u}}$. Then, $E^{(S)}_c = A(S) - E^{(S)}_c \times E_u(S) - P^{(S)}_+ \times A(S)$. Then, $E^{(S)}_c = \frac{A(S)\left(1-P^{(S)}_+\right)}{1+E_u(S)}$ and $E^{(S)}(W) = \frac{1+(W-1)E^{(S)}_c}{(P^{(S)}_+)^{W-1}}$.

We now explain how we numerically approximate the values of $E_u(S)$ and $A(S)$. Let ε be the desired precision. Consider for some value T the difference between $E_u(S)$ and $\sum_{0<t<T} P^{(S)}_{\substack{t\\ u\to u}}$. We have $P^{(S)}_{\substack{t\\ u\to u}} = \prod_{P_q\in S} P^{q}_{\substack{t\\ u\to u}}$ and $P^{q}_{\substack{t\\ u\to u}}$ the probability that a processor that was UP at time 0 is UP at time t without having been $DOWN$. For a processor $P_q\in S$, let $M_q = \begin{bmatrix} P^{(q)}_{u,u} P^{(q)}_{u,r} \\ P^{(q)}_{r,u} P^{(q)}_{r,r} \end{bmatrix}$. Then, $P^{q}_{\substack{t\\ u\to u}} = (M^t_q)[0,0]$. We obtain $P^{q}_{\substack{t\\ u\to u}} = \mu(\lambda^q_1)^t + \nu(\lambda^q_2)^t$ with $\mu,\nu\geq 0$, $\mu+\nu=1$ and $\lambda^q_1 > \lambda^q_2$ eigenvalues of M_q. Then, $P^{q}_{\substack{t\\ u\to u}} \leq (\lambda^q_1)^t$. We obtain $P^{(S)}_{\substack{t\\ u\to u}} \leq \left(\prod_{P_q\in S}\lambda^q_1\right)^t$ and $\sum_{t\geq T} P^{(S)}_{\substack{t\\ u\to u}} \leq \left(\prod_{P_q\in S}\lambda^q_1\right)^T \times \frac{1}{1-\prod_{P_q\in S}\lambda^q_1}$. Let $\Lambda = \prod_{P_q\in S}\lambda^q_1$. We obtain that $T > \frac{\ln(\varepsilon(1-\Lambda))}{\ln(\Lambda)}$ implies $E_u(S) - \sum_{0<t<T} P^{(S)}_{\substack{t\\ u\to u}} \leq \varepsilon$. Thus, we can compute in polynomial time an approximation of $E_u(S)$ at ε in polynomial time.

Similarly, we obtain $A(S) - \sum_{0<t<T} t\times P^{(S)}_{\substack{t\\ u\to u}} \leq \varepsilon$ as soon as $\Lambda^T\left(\frac{T}{1-\Lambda} + \frac{\Lambda}{(1-\Lambda)^2}\right)\leq\varepsilon$. Therefore $A(S)$ can be approximated with precision ε in polynomial time. ■

5.4.2 Probability of success and expected cost of a communication

The same study cannot be conducted concerning communication costs because of the constraint n_{com}. The expected cost of a communication is then estimated as follows. Let S be a set of enrolled processors. For any processor $P_i\in S$, let n_i be the number of time slots of communications needed on this processor to receive the program and all the data of its allocated tasks. Suppose first $|S|\leq n_{com}$. In this particular case, the communication task on each processor can be estimated precisely. On processor P_i, it corresponds to a computation of cost n_i executed on P_i. We obtain an expected time on P_i, $E_i = E^{(P_i)}(n_i)$. We then estimate the expected communication time of the current configuration to

$$E^{(S)}_{comm} = \max_{P_i\in S}\{E^{(P_i)}(n_i)\}\ .$$

With $|S| \geq n_{com}$, the estimation of the communication time is more complicated. In this case, we estimate the communication time as follows:

$$E_{comm}^{(S)} = \max_{P_i \in S} \left\{ \max_{P_i \in S} \left\{ E^{(P_i)}(n_i) \right\}, \frac{\sum_{P_i \in S} n_i}{n_{com}} \right\}$$

This value is used to estimate the probability of success of the communication. We define $P_{ND}^{(P_i)}(t)$ the probability that processor P_i that was UP at time t' does not fail down between time slot t' and $t'+t$. The probability of success is then estimated as

$$P_{comm}^{(S)} = \prod_{P_i \in S} P_{ND}^{(P_i)}(E_{comm}^{(S)}) \ .$$

This values does not take into account the time needed after the end of all communications to obtain that processors become UP simultaneously. The probability of success of an iteration is estimated by multiplying the probability of success of the communications and the probability of success of the computations.

5.5 On-line heuristics

In this section, we propose heuristics to address the on-line version of this scenario. This scenario is more challenging than the previous one, because (i) enrolled processors only make progress in the execution during those time-slots where they are all UP simultaneously; and (ii) the execution advances at the pace of the slowest resource. Hence its seems very unlikely that strategies based on the minimum completion time of individual resources would turn out efficient.

Conceptually, we can envision two main strategies:

Passive. Passive heuristics conservatively keep current processors active as long as possible. In other words, the current configuration is changed only when one of the enrolled processors becomes $DOWN$.

Proactive. Proactive heuristics allow for a complete reconfiguration, possibly terminating ongoing work if a better configuration is determined. The possibility of terminating current tasks makes it possible for an iteration to never complete.A criterion must be derived to decide whether and when such aggressive reconfiguration is worthwhile.

We first propose passive heuristics, and then move to more complex proactive versions. A major difficulty for the latter is the design of a good criterion to decide whether it is worth terminating the current configuration. We proposed several possible criteria and evaluate them experimentally in Section 5.6.

Before going into further details, here follows a summary of the main design rules, together with an intuitive explanation of each one:

- A configuration change is required when one enrolled resource fails. Remember that all previously executed work is lost. However, a processor which has already received one or several communications can reuse them if the scheduler reassigns tasks to it.
- As already pointed out, we have both passive and proactive heuristics. Passive heuristics change the current configuration only when required by a processor crash, while proactive heuristics may change anytime, based upon some evaluation criterion.
- In pro-active heuristics, a heuristic and a criterion are selected. At any time slot, if a new configuration is computed with the selected heuristic. Then, this new configuration is compared with the current one according to the selected criterion. If the new configuration is better than the current

one according to this criterion, a reconfiguration is done and tasks are allocated according to the new configuration. In the other case, the execution continue on the current configuration for an additional time slot, and this operation is executed again.

As in the previous section, all heuristics assign tasks to processors (that must be in the UP state) one-by-one, until m tasks are assigned. The objective of the heuristics is to decide which processors should be used for these tasks.

5.5.1 Pro-active criteria

We propose three criteria to decide whether and when to execute a reconfiguration.

The probability of success of the iterations The probability of success of a new iteration is computed as described in Section 5.4. Concerning the current configuration, if the communication is in progress, the values of remaining time-slots needed to complete the communications n_i are updated according to the communication done, and if the computation is started, the value W of the workload is modified according to the work done.

At any time slot, the configuration selected is the more reliable one. For an heuristic H, the corresponding pro-active heuristic is denoted P-H.

The expected completion time In this case, we select the fastest configuration at any time slot. As in the previous case, at any time slot, the expected cost of the current configuration is recalculated according to the work already done.

For an heuristic H, the corresponding pro-active heuristic is denoted E-H.

The yield This two first criteria does not make a link between reliability and speed of configurations. The first criterion could select slow processors, and the second one unreliable configurations. We use the yield as criterion taking the two parameters into account. The yield is the expected value of the inverse of the global execution time of the current iteration. We estimate this value as follows: For a given configuration of probability of success P, in expected time E, on an iteration running for t time slots, the yield is estimated as

$$Y = \frac{P}{E + t} .$$

The configuration selected is the one that maximize the estimated yield. For an heuristic H, the corresponding pro-active heuristic is denoted Y-H.

Concerning the heuristics, one difficulty is to find the best configuration for one of this criterion. Two classes of heuristics are proposed to solve this problem. In *greedy-indep* heuristics, the values of probability of success and expected time are computed processor by processor as if tasks were independent. The *greedy-coupled* heuristics allocate task by task according to some criterion, even if this choice is not optimal.

The dynamic heuristics below fall into two classes, *greedy-indep* and *greedy-coupled*. Most of these heuristics rely on the assumption that processor availability follows a Markov process, as discussed in Section 5.4.

5.5.2 Greedy-coupled heuristics.

The *greedy-coupled* heuristics aim to find good configurations for a selected criterion, selected among pro-active criteria. However, as tasks are assigned one-by-one, the computed configurations

are not optimal for this criterion.

IP (Incremental:Probability of success)

This heuristic aims configurations with good probability of success. A un-allocated task is assigned on a processor such that the probability of success of assigned tasks is maximal, without taking into account the speed of this processor.

More precisely, considering a set S of processors with at least on task, We compute for any processor P_q the probability of success of the computation: $P^S(q) = P^{S \cup P_q}(W)$ with W the maximum computation cost on a task of $S \cup P_q$ if an additional task is schedule on P_q. We assign the next task to processor P_{q_0} with $q_0 = \text{ArgMax}\left\{P^S(q)\right\}$.

IE (Incremental:Expected completion time)

In this case, we look for fast configurations, without considering its reliability. A un-allocated task is assigned on the processor that minimized the expected execution time of the iteration.

More precisely, considering a set S of processors with at least on task, We compute for any processor P_q the expected completion time of the computation with an additional task on P_q: let m_p the communication time for processor P_p. The communication time is set to
$T^q_{comm} = \max\{\max_{S \cup P_q}\{m_p\}, \frac{1}{n_{com}} \sum S \cup P_q\{m_p\}\}$ The expected computation time T^q_{comp} is computed using results of Section 5.4 We assign the next task to processor P_{q_0} with
$q_0 = \text{ArgMin}\,\{T^q_{comm} + T^q_{comp}\}$.

IY (Incremental:Expected yield)

This heuristic use as criterion the yield of configurations. A new task is allocated on the processor that maximized the yield of the configuration.

Formally, considering a set S of processors with at least on task, We compute for any processor P_q the expected yield of the computation with an additional task on P_q: Let $P^S(q)$ the probability computed for heuristic IP, $T^S(q)$ the expected time computed for heuristic IE and t the time spent since the beginning of the current iteration. We assign the next task to processor P_{q_0} with $q_0 = \text{ArgMax}\left\{\frac{P^S(q)}{t+T^S(q)}\right\}$.

IAY(Incremental:Expected apparent yield)

The yield take into account the time spent on the current iteration. It could seems interesting to only consider the future work and the remaining completion time in stead of the global execution time. The apparent yield is estimated by

$$AY = \frac{P}{E}$$

In this heuristic, we consider the apparent yield.

Formally, considering a set S of processors with at least on task, We compute for any processor P_q the expected yield of the computation with an additional task on P_q: Let $P^S(q)$ the probability computed for heuristic IP, $T^S(q)$ the expected time computed for heuristic IE. We assign the next task to processor P_{q_0} with $q_0 = \text{ArgMax}\left\{\frac{P^S(q)}{T^S(q)}\right\}$.

5.5.3 Greedy-indep heuristics.

We aim to propose heuristics that compute the expected cost on each processor independently of the other processors. This corresponds to the model of Chapter 4. We then re-use heuristics of this chapter. We describe again heuristics for reminder.

MCT (Minimum Completion Time)

For each processor P_q we compute Delay(q), the delay before P_q finishes its current activities if no other processor was used. We then greedily assign each of the remaining $m - m'$ tasks to processors, picking each time the processor with the smallest task completion time. More formally, for each processor P_q, let n_q be the number of tasks already assigned to it (out of the $m - m'$ tasks), and let $CT(P_q, n_q)$ be the estimation of its completion time:

$$CT(P_q, n_q) = \text{Delay}(q) + n_q(T_{\text{data}} + w_q) \ . \tag{5.1}$$

MCT assigns the next task to processor P_{q_0}, where $q_0 = \text{ArgMin}\{CT(P_q, n_q + 1)\}$.
MCT with contention – The estimated completion time in Equation 5.1 does not account for network contention (caused by the master's limited network capacity) during the communication step. Here, we use a simple correcting factor, and replace T_{data} by $\left\lceil \frac{n_{\text{active}}}{n_{com}} \right\rceil T_{\text{data}}$, where n_{active} denotes the number of active processors, i.e., those processors that have been assigned one or several of the $m - m'$ tasks. We derive the new estimation:

$$CT(P_q, n_q) = \text{Delay}(q) + n_q \left(\left\lceil \frac{n_{\text{active}}}{n_{com}} \right\rceil T_{\text{data}} + w_q \right) \ . \tag{5.2}$$

We call MCT* the version of the MCT heuristic that uses the above definition of $CT(P_q, n_q)$.
Expected MCT – Given a workload (i.e., a number of needed time-slots of computation) $CT(P_q, n_q)$, Theorem 5.3 gives the value of $E^{(q)}(CT(P_q, n_q))$, the expected number of time-slots needed for P_q to be UP during $CT(P_q, n_q)$ times-slots without becoming $DOWN$ in between. Using this expectation as the criterion for selecting processors, and depending on whether the correcting factor on T_{data} is used, we obtain one new version of MCT and one new version of MCT*, which we call EMCT and EMCT*, respectively.

LW (Likely to Work)

We build heuristics that consider the probability that a processor P_q, which is UP, will be UP again at least once before becoming $DOWN$. This probability, $P_+^{(q)}$, is given in Chapter 4 and corresponds to $P_+^{(S)}$ with $S = \{P_q\}$ as defined in proof of theorem 5.3. We assign the next task to processor P_{q_0} with the highest probability of being UP for at least the estimated number of needed time-slots to complete its workload, before becoming $DOWN$:

$$q_0 = \text{ArgMax} \left\{ (P_+^{(q)})^{CT(P_q, n_q+1)} \right\} \ .$$

Therefore, we first estimate the size $\mathcal{W}$ of the workload and then the probability that a processor will be in the UP state $\mathcal{W}$ time-slots without becoming $DOWN$ in between. Using Equation 5.2 instead of Equation 5.1, one obtains the LW* heuristic.

UD (Unlikely Down)

Here, we estimate the number $\mathcal{N}$ of time-slots needed for a processor to complete its workload, knowing that it can become $RECLAIMED$. Then we compute the probability that it will not become $DOWN$ for $\mathcal{N}$ time-slots. Given that P_q starts in the UP state, the probability that it does not go to the $DOWN$ state during k time-slots is:

$$P_{UD}^{(q)}(k) = \begin{bmatrix} 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} P_{u,u}^{(q)} & P_{u,r}^{(q)} \\ P_{r,u}^{(q)} & P_{r,r}^{(q)} \end{bmatrix}^{k-1} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

We approximate this expression by forgetting the state of P_q after the first transition:

$$P_{UD}^{(q)}(k) = (1 - P_{u,d}^{(q)}) \left(1 - \frac{P_{u,d}^{(q)} \pi_u^{(q)} + P_{r,d}^{(q)} \pi_r^{(q)}}{\pi_u^{(q)} + \pi_r^{(q)}} \right)^{k-2}.$$

We use this value with $k = E^{(q)}(CT(P_q, n_q + 1))$. UD assigns the next task to the processor P_{q_0} that maximizes the probability of not becoming $DOWN$ before the estimated number of time-slots needed for it to complete its workload, counting the time-slots spent in the $RECLAIMED$ state:

$$q_0 = \text{ArgMax}\{P_{UD}^{(q)}(E^{(q)}(CT(P_q, n_q + 1)))\}.$$

Using Equation 5.2 instead of Equation 5.1, one obtains the UD* heuristic.

5.6 Experiments

We have evaluated the heuristics described in the previous chapter using the same simulator that in the previous section, (available at http://graal.ens-lyon.fr/~fdufosse/changing_platforms.tar.gz).

As in Chapter 4, the quality of an application execution is measured by the time needed to complete 10 iterations, or *makespan*. We have executed all heuristics presented above for several problem instances. For each problem instance, we compute the *degradation from best* (dfb) of each heuristic and we count how often, over all instances, each heuristic is the (or tied with the) best one, so that we can report on numbers of wins for each heuristics. We also compute the standard deviation of the dfb values, and we count the number of solutions at less than 30% of the optimal makespan.

The instances for the experiments are largely similar to those of the previous chapter. All our experiments are for $p = 20$ processors. For each processor P_q, we uniformly pick a random value between 0.90 and 0.99 for each $P_{x,x}^{(q)}$ value (for $x = u, r, d$). We then set $P_{x,y}^{(q)}$ to $0.5 \times (1 - P_{x,x}^{(q)})$, for $x \neq y$. An experimental scenario is defined by the above and by three parameters: n, the number of tasks per iteration, n_{com}, the constraint on the master's communication bandwidth, and the w_{min} parameter. For each processor P_q, we pick w_q uniformly between w_{min} and $10 \times w_{min}$. T_{data} is set to w_{min} and T_{prog} is set to $5 \times w_{min}$. We define experimental scenarios for each of the possible instantiations of (n, n_{com}, w_{min}) given the values shown in Table 5.1.

For $n = 5$, we create 10 random experimental scenario for each possible instantiations of (n_{com}, w_{min}) given the values shown in Table 5.1. For each experimental scenario, we run 10 trials, varying the seed of the random number generator used to determine Markov state transitions. The total number of generated problem instances for this first experiment is 3000.

The results of this first set of experiment is presented in table 5.2. The heuristics are sorted by average degradation from best. The 10 best heuristics for this criteria are then used to a larger experiment, for each values (n, n_{com}, w_{min}) given the values shown in Table 5.1.

Table 5.1: Parameter values for Markov experiments.

parameter	values
p	20
n	$5, 10$
n_{com}	$5, 10, 20$
w_{min}	$1, 2, 3, 4, 5, 6, 7, 8, 9, 10$

5.6.1 Experimental results

Results for $n = 5$. Table 5.2 presents the experimental results of all described heuristics for 5 tasks, sorted by average degradation from best.

The first result of this table is the high values of standard deviations. For any heuristic, this value is higher than the degradation from best, and in some cases, it is more than twice as large than this value (for example heuristics Y-IAY, Y-IY or P-IAY). In most cases, the apparent yield is involved in higher values, but all heuristics have a high standard deviation. The coupled property of the application make the computation very sensitive to failures and unavailability. The apparent yield seems to particularly generate unreliable solutions.

This experiment model however has close similarities with the experiments of Chapter 4. All heuristics described in the previous chapter are used here. The main difference in both results concerns the heuristics MCT and EMCT. For any variant, both results were close in Chapter 4 and however are very different in this case: Heuristic EMCT and its variants are significantly better in this model.

This experiment shows a clear trend for the best results. The expected completion time is the best pro-active criteria, and IE is one of the more efficient heuristics. Heuristics Y-IE and P-IE have the best average degradation from best, with approximately 30%. Heuristics E-IAY and E-IY are a little less efficient for this criteria, but are more often the best heuristic in a simulation with more than 20% of best results. Heuristic E-IAY seems to be the best compromise between this average degradation from best and number of best result. It has however a relatively high standard deviation in comparison with the three other considered heuristics, as all variants of heuristic IAY. We can notice that for any variants, heuristic IAY is better than heuristic IY for the average degradation from best, the number of best results and the number of results with a degradation from best at less than 30%.

This four heuristics are all considering the expected computation time of a computation as a main criteria to select configurations. In practice, three variants of heuristic IE are in the first five heuristics for average degradation from best and, for most of the heuristics, the variant with the expected completion time as pro-active criterion is the most efficient variant. Therefore, the first intuition is that the heuristic E-IE should be very efficient. In practice, the experimental results show that this intuition is wrong and this heuristic has low results, with on average a computation cost two times longer than the optimal, less than 10% of winning instances, and a high standard deviation. In particular, this heuristic has very high execution times for experiments where the fastest processor is unreliable. For similar reasons, heuristics P-IP and Y-IY have low performance.

Algorithm	Average dfb	#wins	#good rate	$stdv$
Y-IE	31.73	17.84	63.05	39.34
P-IE	33.11	16.54	61.90	40.22
E-IAY	34.39	23.49	65.50	54.77
E-IY	43.58	20.30	59.85	66.75
IE	49.04	10.26	63.05	51.30
IAY	55.03	8.85	54.46	78.32

IY	69.02	6.28	48.10	92.75
E-IP	75.67	15.09	45.54	100.06
E-EMCT*	84.10	9.18	40.45	132.24
E-LW	94.08	12.16	39.96	119.80
E-LW*	94.08	12.16	39.96	119.80
E-UD*	94.54	12.86	39.74	119.55
E-UD	97.35	12.34	39.33	123.98
E-EMCT	99.26	5.24	33.01	146.39
E-MCT*	107.24	8.10	35.95	195.38
Y-MCT*	108.22	10.00	41.86	245.04
P-MCT*	108.43	10.00	41.56	244.88
IP	110.86	4.57	35.02	139.20
E-IE	112.06	9.74	34.91	245.57
EMCT*	117.99	3.64	36.21	170.18
Y-IAY	122.91	19.33	54.46	414.19
LW*	128.16	3.64	30.37	151.34
LW	129.85	3.61	30.37	152.49
EMCT	134.12	1.60	28.92	182.31
UD*	137.84	3.61	30.93	158.62
UD	139.01	3.49	30.33	159.42
Y-IY	139.99	15.87	48.10	445.53
P-IAY	154.23	16.36	49.55	525.96
Y-UD*	157.81	9.29	30.93	265.64
Y-UD	161.32	8.88	30.33	273.22
Y-LW*	169.86	9.48	30.37	311.94
Y-LW	169.86	9.48	30.37	311.94
Y-EMCT*	171.15	7.43	36.21	487.99
P-UD*	178.35	8.51	28.96	305.94
P-UD	180.83	8.14	28.77	308.63
P-IY	181.55	13.31	40.19	549.40
Y-IP	182.56	12.01	35.02	469.33
P-EMCT*	188.02	7.47	34.72	540.64
Y-EMCT	189.66	4.16	28.92	503.38
MCT*	190.32	1.93	41.86	340.63
E-MCT	190.36	1.71	14.01	255.80
Y-MCT	194.49	2.34	18.25	360.79
P-MCT	195.52	2.23	17.73	360.65
P-LW*	196.02	8.96	28.92	363.72
P-LW	196.02	8.96	28.92	363.72
P-EMCT	204.75	4.05	27.58	547.80
P-IP	221.83	11.49	33.23	594.46
MCT	325.17	0.56	18.25	471.36

Table 5.2: Comparison of heuristics performance with 5 tasks

Comparison of the 10 best heuristics. The 10 best heuristics for the average degradation from best in the previous experiment are the following: Y-IE, P-IE, E-IAY, E-IY, IE , IAY, IY, E-IP,

E-EMCT* and E-LW.

Only two of these heuristics are not considering the expected execution time as a main criteria: heuristics IAY and IY. This two heuristics are efficient in term of average degradation from best, but have low results in term of best results. They obtain the lower value of these 10 heuristics for this criterion.

We first present a general Table 5.2 presenting the results on the global experiment, then Tables 5.3 and 5.4 respectively present the results in the experiment with 5 and 10 tasks.

In the previous experiment, we considered that the heuristic E-IAY was a good compromise between average degradation from best and number of best heuristic of an instance. This is true for heuristic of 5 tasks, but for 10 tasks, this heuristic is the best one for both criteria.

Heuristics Y-IE and P-IE however obtain good results in experiments with 10 tasks. We notice that in this last experiment, if heuristic Y-IE has a smaller average degradation from best than heuristic P-IE, it has a lower number of best results.

Algorithm	Average dfb	#wins	#good rate	$stdv$
Y-IE	33.06	17.76	69.71	40.33
P-IE	34.48	16.66	68.02	41.34
E-IAY	35.26	24.83	71.37	55.80
E-IY	45.44	20.38	64.22	69.38
IE	51.40	10.81	69.71	59.83
IAY	59.32	8.46	70.12	93.12
IY	74.69	6.02	63.08	106.61
E-IP	77.08	15.41	49.51	103.19
E-EMCT*	92.23	8.63	43.14	164.71
E-LW*	92.80	12.65	44.65	123.30

Figure 5.2: General results for the 10 best heuristics

Algorithm	Average dfb	#wins	#good rate	$stdv$
Y-IE	32.30	17.69	70.44	40.21
P-IE	33.83	16.36	68.67	41.19
E-IAY	35.34	23.74	70.99	56.98
E-IY	44.10	20.48	64.86	67.89
IE	47.59	11.33	70.44	50.82
IAY	57.57	9.18	69.66	96.75
IY	71.02	6.63	63.67	108.21
E-IP	73.82	15.68	50.17	98.56
E-EMCT*	87.23	9.32	44.80	162.71
E-LW	90.68	12.82	45.07	119.19

Figure 5.3: Results with 5 tasks for the 10 best heuristics

Finally, the heuristics IAY and IY had already few best results with 5 tasks. This values further decrease in the experiment with 10 tasks.

Algorithm	Average dfb	#wins	#good rate	$stdv$
E-IAY	34.83	31.20	73.60	48.23
Y-IE	37.48	18.20	65.40	40.98
P-IE	38.31	18.40	64.20	42.21
E-IY	53.32	19.80	60.40	77.59
IAY	69.62	4.20	72.80	67.90
IE	73.74	7.80	65.40	97.15
E-IP	96.20	13.80	45.60	127.04
IY	96.29	2.40	59.60	96.62
E-LW	105.30	11.60	42.20	145.12
E-EMCT*	121.64	4.60	33.40	175.99

Figure 5.4: Results with 10 tasks for the 10 best heuristics

5.7 Conclusion

This chapter is a follow-on of Chapter 4. The platform model is the same, but the application model differs. While tasks were independent in the previous chapter, they are now tightly coupled. This modification makes the computations more sensitive to failures. With this model, a single failure of one processor can annihilate a long on-going computation on many processors.

We have proved the NP-completeness of the off-line problem and provided optimal algorithms for polynomial particular instances. By assuming a Markov model of processors availability, we have proposed polynomial time approximation schemes to compute the expected completion time of a computation and its probability of success. The heuristics of Chapter 4 have been adapted to this new application model, and new heuristics were derived from the probability results. For any heuristic, pro-active variants were derived, using as criteria the expected completion time, the probability of success and the expected yield of the platform.

The simulations have shown that for each heuristic, performance varies to a great extent from one experiment to another. As a result, all heuristics have high standard deviations. However a few set of heuristics was significantly better than the others, and in particularly, heuristic E-IAY has good average performance for small instances, and is clearly better than the other heuristics for larger ones. We conclude that maximizing the apparent yield seems to be the best approach, particularly in a proactive setting.

Chapter 6

Conclusion and perspectives

6.1 Conclusion

In this thesis, we have explored many scheduling problems with reliability as main criterion. We aimed at proving the theoretical complexity of the various instances, and for NP-complete problems, at finding approximation results and providing heuristics. In the following we detail our contributions.

Mapping filtering streaming applications

Our first contribution is a theoretical study on the problem of mapping filtering streaming applications on homogeneous and heterogeneous platforms.

First, we have considered one-to-one mappings. In a simplified model without communication cost, we have exhibited polynomial time algorithms for latency and period optimization problems on homogeneous platforms, and we have proved the NP-hardness of these problems on heterogeneous platforms. Proofs of the inapproximability of these instances (unless $P = NP$) were provided. Then we have identified three natural and realistic communication models, with and without communication/computation overlap, and with one-port or bounded multi-port communications. We have been able to provide the complexity of all the optimization problems under study.

Then, we have extended this work to general mappings on heterogeneous linear platforms. We have dealt with different instances for period and latency optimization problems, with proportional or arbitrary computation costs, and without or with communication costs.

All these results have been published in [92, 93, 94, 91].

Reliability and performance optimization of pipelined real-time systems

In this chapter, whose published version is [96], we have addressed problems related to the mapping of linear workflows on homogeneous and heterogeneous distributed platforms. The main goal was to optimize the reliability of the mapping through task replication, while enforcing bounds on performance-oriented criteria (period and latency). We derived a comprehensive set of NP-hardness complexity results, together with optimal algorithms for polynomial instances. Altogether, these results provide a solid theoretical foundation for the study of multi-criteria mappings of linear workflows. Another contribution of this chapter is the introduction of a realistic communication model that nicely accounts for the inherent physical limitations on the communication capabilities of state-of-the-art processors.

On homogeneous platforms, an integer linear program has been presented to solve the problem of maximizing the reliability with bounds on period and on latency, while polynomial-time heuristics

were derived for the most general problems. We have proposed two heuristics: HEUR-L that attempts to minimize the latency and HEUR-P that attempts to minimize the period. Our experiments have demonstrated the efficiency of the heuristics, and the supremacy of HEUR-P in most cases.

Scheduling parallel iterative applications on volatile resources

This study has addressed the problem of scheduling iterative applications with independent tasks on desktop grids. Processors are subject to failures and are likely to be reclaimed by their owners. We have modeled communications by considering bandwidth constraints in a master-worker schedule. Data needed by tasks were divided in data common to all tasks, and individual data specific for any task. This work has been published in [97].

In this context, we have studied the theoretical complexity of the off-line problem. We have provided a proof of NP-completeness and an inapproximability result for the general problem, as well as an algorithm for a polynomial particular instance.

The on-line study of this problem has assumed a Markov behavior of processors states. Closed-form formulas were given to compute the expected computation times and probabilities of success of computations.

A set of heuristics was provided based on random decisions, or on the different theoretical formulas obtained using the Markov assumption. A large set of simulations with the Markov model was used to compare these heuristics.

Scheduling parallel iterative coupled applications on volatile resources

This chapter extends the previous results to tightly coupled tasks. In this case, tasks have to progress at same speed. This constraint makes the schedule more challenging. A single failure of one processor can annihilate a long on-going computation on many processors. As in the previous chapter, the off-line problem has been theoretically studied. NP-completeness was proved for instances with and without contention constraints. Optimal algorithms were provided for some polynomial instances.

Closed-form formula could not be presented for probability results, contrarily to the previous chapter. However, polynomial-time approximation schemes were provided for expected computation times and success probabilities. The yield could only be imprecisely estimated. A variant to the yield, named apparent yield, was considered. Instead of considering the expected execution time of the whole iteration, it only considers, at each time-step the expected remaining time until the end of the computations.

A set of heuristics was provided, some coming from the previous chapter, and others based on probability results. Pro-active variants were generated based on the previous heuristics, and configuration changes were conducted according to various criteria. Simulations were used to compare these heuristics.

6.2 Perspectives

These studies can be extended by many way. In the following, for any chapter, many directions are provided for future work, and some general perspectives are then detailed.

Mapping filtering streaming applications

In the first chapter, a complete set of complexity results was presented for all the instances. However, few approximation results were provided for NP-complete instances. Many problems for one-to-

one mappings and general mappings should be studied, aiming at approximation results. In addition, designing an integer linear program would also be interesting for these instances.

In [100], greedy heuristics have been derived for one-to-one mappings without communication costs, and compared to the optimal solution. The latter was computed using an integer linear program. The goal was to evaluate the respective impacts of the different parameters of the problem. These heuristics should be adjusted for the different communication models, and more elaborate heuristics should be derived.

For polynomial instances, many criteria can be studied in addition to period and latency. We could for example take into account the reliability of computations or the energy consumption.

Reliability and performance optimization of pipelined real-time systems

In this part, we have considered the optimization of the reliability in the scheduling problem of linear workflows. The communications were transmitted through routing operations. This communication model makes the computation of success probability simpler. A theoretical study should be conducted without this hypothesis. In addition, this chapter has only considered linear workflows. The problem could be extended to general workflows.

As in the previous chapter, approximation results were incomplete and should be extended. An integer linear program should be searched for the problem on heterogeneous platforms. Two heuristics were proposed and compared using simulations. A larger set of heuristics could be proposed and compared to the currently available ones.

The probability law has not considered the aging of processors. This criterion, however, influences the occurrences of transient failures. A more accurate probability law, taking into account this aging, could be considered.

Scheduling parallel iterative applications on volatile resources

The main weakness of this study is the low accuracy of the Markov hypothesis for modeling real life traces. More realistic probabilistic models have been proposed. These probabilistic distributions should be used to generate new formula for expected time and success probability of computations, and new heuristics should be proposed and compared with the ones presented here, all this using real life traces. However, real life traces and distribution models often only handle only two states UP and $DOWN$.

In the set of provided heuristics, some only considered an estimation of the execution time, and others only aimed an estimation of the probability of success. A criterion merging these two objectives could be derived, with expected good complexity results for both small and large instances. The yield of the platform could be an interesting criterion, under the condition of being able to provide a closed-form formula for its value. A study of the two state model, with only available and down processor states, should be considered with both an on-line and off-line approach.

Scheduling parallel iterative coupled applications on volatile resources

In this study, contrarily to the previous chapter, the off-line study has not led to any approximation result. It would be interesting to derive some approximation result, or some integer linear program, for this model.

As above, more accurate distribution laws could be used to compare the heuristics, and to propose new ones. Pareto distribution and Weibull distribution are usually considered as the more appropriate, however they are heavy-tail distributions, and the resulting standard deviations are distorted by extreme

values. In such a model with large standard deviation, even for the Markov distribution, the consistency of simulation results would be difficult to assess.

The estimation of the yield given in this chapter is not accurate. A closed-form formula or an approximation value would be desirable. New heuristics could result from such a formula or approximated value. Finally, as in the previous chapter, a study of the two state model, with only available and down processor states, should be undertaken.

General perspectives

In this thesis, the main method used to improve schedule reliability is *replication*. However, this approach is very expensive in CPU resource and in energy consumption. The criterion of energy consumption could be accounted for, in all the previous studies, in addition to reliability. This would enable us to obtain more realistic schedules in term of costs. A preliminary work [98] has been conducted in this area. However, this work only aims at minimizing the energy consumption, without taking reliability into account.

Many other methods can be used to increase the reliability of schedules, as checkpointing or migration. Such methods could be applied to our different models, and compared to our results using replication, in term of reliability and of energy consumption.

These three methods (replication, checkpointing, migration) are however not conflicting. We could address problems aiming at optimizing the reliability, using any combination of these methods. The respective advantages and drawbacks of each method would be assessed and compared, with the goal of establishing complexity and probability results, and of providing optimal algorithms, approximations or efficient heuristics.

Appendix A

Bibliography

[1] Alessandro Agnetis, Paolo Detti, Marco Pranzo, and Manbir S. Sodhi. Sequencing unreliable jobs on parallel machines. *Journal of Scheduling*, 2008.

[2] Cosimo Anglano, John Brevik, Massimo Canonico, Dan Nurmi, and Rich Wolski. Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids. In *Proceedings of Grid Computing*, pages 56–63, 2006.

[3] Ismail Assayad, Alain Girault, and Hamoudi Kalla. A bi-criteria scheduling heuristics for distributed embedded systems under reliability and real-time constraints. In *International Conference on Dependable Systems and Networks, DSN'04*, pages 347–356, Firenze, Italy, June 2004. IEEE.

[4] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January 2004.

[5] Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive ordering of pipelined stream filters. In *SIGMOD'04: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 407–418. ACM Press, 2004.

[6] Jacques M. Bahi, Sylvain Contassot-Vivier, and Raphael Couturier. *Parallel Iterative Algorithms: From Sequential to Grid Computing*. Chapman and Hall/CRC Press, 2007.

[7] Harlod S. Balaban. Some effects of redundancy on system reliability. In *National Symposium on Reliability and Quality Control*, pages 385–402, Washington (DC), USA, January 1960.

[8] Massimo Baleani, Alberto Ferrari, Leonardo Mangeruca, Maurizio Peri, Saverio Pezzini, and Aalberto Sangiovanni-Vincentelli. Fault-tolerant platforms for automotive safety-critical applications. In *International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES'03*, San Jose (CA), USA, November 2003. ACM.

[9] Anne Benoit, Veronika Rehn-Sonigo, and Yves Robert. Optimizing latency and reliability of pipeline workflow applications. In *HCW'08, the 17th Heterogeneity in Computing Workshop*. IEEE Computer Society Press, 2008.

[10] Anne Benoit and Yves Robert. Mapping pipeline skeletons onto heterogeneous platforms. *Journal of Parallel and Distributed Computing*, 68(6):790–808, 2008.

[11] Anne Benoit and Yves Robert. Complexity results for throughput and latency optimization of replicated and data-parallel workflows. *Algorithmica*, 57(4):689–724, 2010.

[12] Prashanth B. Bhat, Cauligi S. Raghavendra, and Viktor K. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63:251–263, 2003.

[13] BOINC: Berkeley Open Infrastructure for Network Computing. http://boinc.berkeley.edu.

[14] George Bosilca, Aurelien Bouteiller, Franck Cappello, Samir Djilali, Gilles Fedak, Cecile Germain, Thomas Herault, Pierre Lemarinier, Oleg Lodygensky, Frederic Magniette, Vincent Neri, and Anton Selikhov. Mpich-v: toward a scalable fault tolerant mpi for volatile nodes. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Supercomputing 2002, pages 1–18, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[15] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 3rd edition, 2001.

[16] Jen Burge, Kamesh Munagala, and Utkarsh Srivastava. Ordering pipelined query operators with precedence constraints. Research Report 2005-40, Stanford University, November 2005.

[17] EunJoung Byun, SungJin Choi, MaengSoon Baik, JoonMin Gil, ChanYeol Park, and ChongSun Hwang. MJSA: Markov job scheduler based on availability in desktop grid computing environment. *Future Generation Computer Systems*, 23(4):616–622, 2007.

[18] Surajit Chaudhuri and Kyuseok Shim. Optimization of queries with user-defined predicates. *ACM Transactions on Database Systems*, 24(2):177–228, 1999.

[19] Andrew A. Chien, Brad Calder, Stephen Elbert, and Karan Bhatia. Entropia: Architecture and performance of an enterprise desktop grid system. *Journal of Parallel and Distributed Computing*, 63:597–610, 2003.

[20] CPLEX. ILOG CPLEX: High-performance software for mathematical programming and optimization. http://www.ilog.com/products/cplex/.

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

[22] Milind. Dawande, Pinar Keskinocak, Jayaminathan M. Swaminathan, and Sridhar Tayur. On bipartite and multipartite clique problems. *Journal of Algorithms*, 41:388–403, 2001.

[23] Josemar Rodriguez de Souza, Eduardo Argollo, Angelo Duarte, Dolores Rexachs, and Emilio Luque. Fault tolerant master-worker over a multi-cluster architecture. In *Proceedings of ParCo 2005*, pages 465–472. NIC Series, Vol. 33, 2006.

[24] Atakan Dogan and Füsun Özgüner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):308–323, March 2002.

[25] Jack Dongarra, Emmanuel Jeannot, Erik Saule, and Zhiao Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 280–288. ACM Press, 2007.

[26] Fanny Dufossé. Source Code for the Heuristics. http://graal.ens-lyon.fr/~fdufosse/filters/.

[27] Trilce Estrada, David Flores, Michela Taufer, Patricia Teller, Andre Kerstens, and David Anderson. The Effectiveness of Threshold-Based Scheduling Policies in BOINC Projects. In *Proceedings of e-Science'06*, 2006.

[28] Trilce Estrada, Olac Fuentes, and Michela Taufer. A distributed evolutionary method to design scheduling policies for volunteer computing. *ACM SIGMETRICS Performance Evaluation Review*, 36(3):40–49, 2008.

[29] Graham E. Fagg and Jack Dongarra. FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World. In *Proceedings of the 7th EuroPVM/MPI*, pages 346–353. Springer-Verlag, 2000.

[30] Christian Ferdinand, Reinhold Heckmann, Marc Langenbach, Florian Martin, Michael Schmidt, Henrik Theiling, Stephan Thesing, and Reinhard Wilhelm. Reliable and precise WCET determination for a real-life processor. In *International Workshop on Embedded Software, EMSOFT'01*, volume 2211 of *LNCS*. Springer-Verlag, 2001.

[31] Daniela Florescu, Andreas Grunhagen, and Donald Kossmann. Xl: A platform for web services. In *CIDR 2003, First Biennial Conference on Innovative Data Systems Research*, 2003.

[32] Noriyuki Fujimoto and Kenichi Hagihara. Near-Optimal Dynamic Task Scheduling of Independent Coarse-Grained Tasks onto a Computational Grid. In *Proceedings of the 32nd International Conference on Parallel Processing (ICPP'2003)*, 2003.

[33] Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvari. Multi-criteria reinforcement learning, 1998.

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

[35] Felix Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, March 1999.

[36] Alain Girault and Hamoudi Kalla. A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. *IEEE Transactions on Dependable and Secure Computing*, 6(4):241–254, December 2009.

[37] Alain Girault, Eric Saule, and Denis Trystram. Reliability versus performance for critical applications. *Journal of Parallel and Distributed Computing*, 69(3):326–336, March 2009.

[38] William Gropp. MPICH2: A New Start for MPI Implementations. In *PVM/MPI*, page 7, 2002.

[39] Rachid Guerraoui and André Schiper. Software-Based Replication for Fault Tolerance. *IEEE Computer*, 30:68–74, 1997.

[40] Mourad Hakem and Franck Butelle. A bi-objective algorithm for scheduling parallel applications on heterogeneous systems subject to failures. In *Rencontres Francophones du Parallélisme, RENPAR'06*, Perpignan, France, October 2006.

[41] Stephen L. Hary and Füsun Özgüner. Precedence-constrained task allocation onto point-to-point networks for pipelined execution. *IEEE Trans. Parallel and Distributed Systems*, 10(8):838–851, 1999.

[42] Johan Håstad. Some optimal inapproximability results. In *STOC '97*, pages 1–10. ACM, 1997.

[43] Haiwu He, Gilles Fedak, Bing Tang, and Franck Cappello. BLAST Application with Data-Aware Desktop Grid Middleware. In *Proceedings of CCGrid*, pages 284–291, 2009.

[44] Abdelsalam Heddaya and Kihong Park. Mapping parallel iterative algorithms onto workstation networks. In *HPDC'94*, pages 211 –218, 1994.

[45] Eric Heien, David Anderson, and Kenichi Hagihara. Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments. *Journal of Grid Computing*, 7(4):501–518, 2009.

[46] Joseph M. Hellerstein. Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 267–276, 1993.

[47] Bo Hong and Viktor K. Prasanna. Bandwidth-aware resource allocation for heterogeneous computing systems to maximize throughput. In *Proceedings of the 32nd International Conference on Parallel Processing (ICPP'2003)*. IEEE Computer Society Press, 2003.

[48] Bo Hong and Viktor K. Prasanna. Adaptive allocation of independent tasks to maximize throughput. *IEEE Transactions on Parallel and Distributed Systems*, 18(10):1420–1435, 2007.

[49] Bahman Javadi, Derrick Kondo, Jean-Marc Vincent, and David Anderson. Mining for Statistical Models of Availability in Large-Scale Distributed Systems: An Empirical Study of SETI@home. In *Proceedings of the 17th MASCOTS*, 2009.

[50] Emmanuel Jeannot, Erik Saule, and Denis Trystram. Bi-objective approximation scheme for makespan and reliability optimization on uniform parallel machines. In *Euro-Par*, volume 5168 of *Lecture Notes in Computer Science*, pages 877–886. Springer, 2008.

[51] Paul A. Jensen and Mandell Bellmore. An algorithm to determine the reliability of a complex system. *IEEE Transactions on Reliability*, 18:169–174, November 1969.

[52] John C. Knight and Nancy G. Leveson. An experimental evaluation of the assumption of independence in multi-version programming. *IEEE Transactions on Software Engineering*, 12(1):96–109, 1986.

[53] Derrick Kondo, Andrew A. Chien, and Henri Casanova. Resource Management for Rapid Application Turnaround on Enterprise Desktop Grids. In *Proceedings of SC'04*, 2004.

[54] Troy Leblanc, Rakhi Anand, Edgar Gabriel, and Jaspal Subhlok. VolpexMPI: An MPI Library for Execution of Parallel Applications on Volatile Nodes. In *Proc. of EuroPVM/MPI 2009*, pages 124–133. Springer-Verlag, 2009.

[55] Arnaud Legrand, Helene Renard, Yves Robert, and Frederic Vivien. Mapping and load-balancing iterative computations on heterogeneous clusters with shared links. *IEEE Transactions on Parallel and Distributed Systems*, 15:546–558, 2004.

[56] David Lloyd and Myron Lipow. *Reliability: Management, Methods, and Mathematics*, chapter 9. PH, 1962.

[57] George Mavrotas. Effective implementation of the [epsilon]-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2):455 – 465, 2009.

[58] Christopher Moretti, Timothy Faltemier, Douglas Thain, and Patrick Flynn. Challenges in Executing Data Intensive Biometric Workloads on a Desktop Grid. In *Proceedings of PCGrid*, 2007.

[59] Daniel Nurmi, John Brevik, and Rich Wolski. Modeling Machine Availability in Enterprise and Wide-area Distributed Computing Environments. In *Proceedings of Europar*, 2005.

[60] Mourad Ouzzani and Athman Bouguettaya. Query processing and optimization on the web. *Distributed and Parallel Databases*, 15(3):187–218, 2004.

[61] Ali Pinar and Cevdet Aykanat. Fast optimal load balancing algorithms for 1D partitioning. *Journal of Parallel and Distributed Computing*, 64(8):974–996, 2004.

[62] Michele Pizza, Lorenzo Strigini, Andrea Bondavalli, and Felicita Di Giandomenico. Optimal discrimination between transient and permanent faults. In *In Third IEEE International High-Assurance Systems Engineering Symposium*, pages 214–223, 1998.

[63] Paul Pop, Kåre Poulsen, Viacheslav Izosimov, and Petru Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *International Conference on Hardware-Software Codesign and System Synthesis, CODES+ISSS'07*, Salzburg, Austria, October 2007. ACM.

[64] Xiaojuan Ren, Seyong Lee, Rudolf Eigenmann, and Saurabh Bagchi. Prediction of Resource Availability in Fine-Grained Cycle Sharing Systems Empirical Evaluation. *Journal of Grid Computing*, 5(2):173–195, 2007.

[65] Günter Rudolph. On a multi-objective evolutionary algorithm and its convergence to the pareto set. In *proceedings of the 5th IEEE conference on evolutionary computation*, pages 511–516. IEEE Press, 1998.

[66] Erik Saule and Denis Trystram. Analyzing scheduling with transient failures. *Information Processing Letters*, 109(11):539–542, 2009.

[67] Sol M. Shatz and Jia-Ping Wang. Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Transactions on Reliability*, 38(1):16–26, April 1989.

[68] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI the complete reference*. The MIT Press, 1996.

[69] Jean Souyris, Erwan Le Pavec, Guillaume Himbert, Victor Jégu, Guillaume Borios, and Reinhold Heckmann. Computing the worst case execution time of an avionics program by abstract interpretation. In *International Workshop on Worst-case Execution Time, WCET'05*, pages 21–24, Mallorca, Spain, July 2005.

[70] Utkarsh Srivastava, Kamesh Munagala, and Jennifer Widom. Operator placement for in-network stream query processing. In *PODS'05: Proceedings of the 24th ACM Symposium on Principles of Database Systems*, pages 250–258. ACM Press, 2005.

[71] Utkarsh Srivastava, Kamesh Munagala, Jennifer Widom, and Rajeev Motwani. Query optimization over web services. In *VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 355–366. VLDB Endowment, 2006.

[72] Paul Stelling, Cheryl DeMatteis, Ian Foster, Carl Kesselman, Craig Lee, and Gregor von Laszewski. A fault detection service for wide area distributed computations. *Cluster Computing*, 2(2):117–128, 1999.

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

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

[75] Kenjiro Taura and Andrew A. Chien. A heuristic algorithm for mapping communicating tasks on heterogeneous resources. In *Heterogeneous Computing Workshop*, pages 102–115. IEEE Computer Society Press, 2000.

[76] Toshiaki Toyama, Yoshito Yamada, and Katsumi Konishi. A Resource Management System for Data-Intensive Applications in Desktop Grid Environments. In *Proceedings of PDCS*, 2006.

[77] Nagavijayalakshmi Vydyanathan, Umit Catalyurek, Tashin Kurc, P. Saddayappan, and Joel 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, January 2007. Short version appears in EuroPar'2008.

[78] Nagavijayalakshmi Vydyanathan, Umit Catalyurek, Tashin Kurc, P. Saddayappan, and Joel Saltz. Toward optimizing latency under throughput constraints for application workflows on clusters. In *Euro-Par'07*, LNCS 4641, pages 173–183. Springer Verlag, 2007.

[79] Nagavijayalakshmi Vydyanathan, Umit Catalyurek, Tashin Kurc, P. Saddayappan, and Joel Saltz. A duplication based algorithm for optimizing latency under throughput constraints for streaming workflows. In *Proceedings of the 37th International Conference on Parallel Processing (ICPP'2008)*, pages 254–261. IEEE Computer Society Press, 2008.

[80] Nagavijayalakshmi Vydyanathan, Umit Catalyurek, Tashin Kurc, P. Saddayappan, and Joel Saltz. Optimizing latency and throughput of application workflows on clusters. Research Report OSU-CISRC-4/08-TR17, Ohio State University, Columbus, OH, April 2008.

[81] Joshua Wingstrom and Henri Casanova. Probabilistic Allocation of Tasks on Desktop Grids. In *Proceedings of PCGrid*, 2008.

[82] Rich Wolski, Daniel Nurmi, and John Brevik. An analysis of availability distributions in condor. In *Proceedings of the IPDPS Workshop on Next-Generation Software*, 2007.

[83] Qishi Wu, Jinzhu Gao, Mengxia Zhu, Nageswara S.V. Rao, Jian Huang, and Sitharama S. Iyengar. Self-adaptive configuration of visualization pipeline over wide-area networks. *IEEE Transactions on Computers*, 57(1):55–68, 2008.

[84] Qishi Wu and Yi Gu. Supporting distributed application workflows in heterogeneous computing environments. In *14th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE Computer Society Press, 2008.

[85] Wenci Yu. *The two-machine flow shop problem with delays and the one-machine total tardiness problem*. PhD thesis, Technishe Universiteit Eidhoven, June 1996.

[86] Wenci Yu, Han Hoogeveen, and Jan Karel Lenstra. Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5):333–348, 2004.

[87] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z Sheng. Quality driven web services composition. In *Proceedings of the 12nd international conference on World Wide Web*, WWW '03, pages 411–421, New York, NY, USA, 2003. ACM.

[88] Hongyu Zhang. On the distribution of software faults. *IEEE Transactions on Software Engineering*, 34:301–302, March 2008.

[89] Dayi Zhou and Virginia Lo. Wave Scheduler: Scheduling for Faster Turnaround Time in Peer-based Desktop Grid Systems. In *Proceedings of the 11th JSSPP Workshop*, 2005.

[90] Dakai Zhu, Rami Melhem, and Daniel Mossé. The effects of energy management on reliability in real-time embedded systems. In *International Conference on Computer Aided Design, ICCAD'04*, pages 35–40, San Jose (CA), USA, November 2004.

Appendix B

Publications

Articles in international refereed journals and book chapter

[91] Kunal Agrawal, Anne Benoit, Fanny Dufossé, and Yves Robert. Mapping filtering streaming applications. *Algorithmica*, 2010.

Articles in international refereed conferences

[92] Anne Benoit, Fanny Dufossé, and Yves Robert. Filter placement on a pipelined architecture. In *11th Workshop on Advances in Parallel and Distributed Computational Models APDCM 2009*. IEEE Computer Society Press, 2009.

[93] Anne Benoit, Fanny Dufossé, and Yves Robert. On the complexity of mapping pipelined filtering services on heterogeneous platforms. In *IPDPS'2009, the 23rd IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society Press, 2009.

[94] Kunal Agrawal, Anne Benoit, Fanny Dufossé, and Yves Robert. Mapping filtering streaming applications with communication costs. In *21st ACM Symposium on Parallelism in Algorithms and Architectures SPAA 2009*. ACM Press, 2009.

[95] Anne Benoit, Bruno Gaujal, Fanny Dufossé, Matthieu Gallet, and Yves Robert. Computing the throughput of probabilistic and replicated streaming applications. In *22nd ACM Symposium on Parallelism in Algorithms and Architectures SPAA 2010*. ACM Press, 2010.

[96] Anne Benoit, Fanny Dufossé, Alain Girault, and Yves Robert. Reliability and performance optimization of pipelined real-time systems. In *International Conference on Parallel Processing*, page 20, 2010.

[97] Henri Casanova, Fanny Dufossé, Yves Robert, and Frédéric Vivien. Scheduling parallel iterative applications on volatile resources. In *IPDPS'2011, the 25th IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society Press, 2011.

[98] Guillaume Aupy, Anne Benoit, Fanny Dufossé, and Yves Robert. Brief announcement: Reclaiming the energy of a schedule, models and algorithms. In *23rd ACM Symposium on Parallelism in Algorithms and Architectures SPAA 2011*. ACM Press, 2011.

Research reports

[99] Anne Benoit, Fanny Dufossé, and Yves Robert. Mapping filter services on heterogeneous platforms. Research Report RR-2008-19, LIP, ENS Lyon, June 2008.

[100] Anne Benoit, Fanny Dufossé, and Yves Robert. On the complexity of mapping pipelined filtering services on heterogeneous platforms. Research Report RR-2008-30, LIP, ENS Lyon, October 2008.

[101] Kunal Agrawal, Anne Benoit, Fanny Dufossé, and Yves Robert. Mapping Filtering Streaming Applications With Communication Costs. Research Report RR-2009-06, LIP, ENS Lyon, February 2009.

[102] Anne Benoit, Fanny Dufossé, Matthieu Gallet, Bruno Gaujal, and Yves Robert. Computing the throughput of probabilistic and replicated streaming applications. Research Report RR-7182, INRIA, January 2010.

[103] Henri Casanova, Fanny Dufossé, Yves Robert, and Frédéric Vivien. Scheduling Parallel Iterative Applications on Volatile Resources. Research Report RR-2010-31, LIP, ENS Lyon, October 2010.

[104] Anne Benoit, Fanny Dufossé, Alain Girault, and Yves Robert. Reliability and performance optimization of pipelined real-time systems. Research Report RR-7509, INRIA, January 2011.

[105] Guillaume Aupy, Anne Benoit, Fanny Dufossé, and Yves Robert. Reclaiming the energy of a schedule: models and algorithms. Research Report RR-7598, INRIA, April 2011.

Résumé :
Les travaux présentés dans cette thèse portent sur le placement et l'ordonnancement d'applications de flots de données. On se place dans le contexte de plates-formes instables, composées de processeurs sujets à des pannes.
Dans une première partie, on considère un type particulier d'applications de flots de données : les services filtrants. Un service filtrant est une tâche qui modifie la taille du fichier qu'elle doit traiter. Dans ce contexte, on veut obtenir rapidement le résultat de chaque calcul (et donc minimiser la latence) et traiter autant de calculs que possible par unité de temps (et donc minimiser la période). On néglige d'abord les coûts de communications. On considère alors les problèmes uni-critères et bi-critères sur des plates-formes homogènes et hétérogènes. Puis plusieurs modèles de communications sont proposés, et leur impact sur l'ordonnancement d'un ensemble de service filtrant est étudié. On considère enfin l'ordonnancement d'un tel calcul sur une chaîne de processeurs. La complexité de chaque variante de ce problème est démontrée.
Le comportement d'un service filtrant est comparable à celui d'un calcul effectué sur un processeur non fiable : certains résultats vont être calculés, et d'autres seront perdus. On considère donc le modèle de panne le plus fréquemment rencontré parmi les processeurs modernes : les pannes transitoires. Ce type de panne perturbe le fonctionnement d'un processeur pendant un très court laps de temps. Ces pannes sont souvent occasionnées par des baisses de tension électrique. On considère que ces pannes sont instantanées, et ne perturbent que le calcul en cours. On veut donc effectuer un calcul à la fois fiable, et efficace. Trois critères sont donc considérés : la période, la latence et la probabilité de succès du calcul. La probabilité d'apparition de pannes est supposée constante au cours du temps. La complexité de chaque variante de ce problème est démontrée. Nous proposons deux heuristiques, dont les performances respectives sont comparées expérimentalement.
Si les pannes transitoires sont les pannes les plus fréquemment rencontrées sur des grilles de calculs classiques, certains types de plates-formes de calcul rencontrent d'autres types de défaillances. Les grilles de volontaires, en particulier, sont extrêmement instables. Les grilles de volontaires sont des ensembles de machines que des propriétaires mettent à disposition tant qu'ils ne les utilisent pas. Chaque machine peut donc à tout moment être éteinte par son propriétaire. Sur ce type de plate-forme, on veut exécuter des calculs iteratifs : Une application iterative est un ensemble de tâches qui doit être exécuté à plusieurs reprises. Toutes les tâches sont exécutées, puis les processeurs se synchronisent, et on exécute à nouveau les tâches, et ainsi de suite. Deux variantes de ce problème sont considérées : une application constituée de tâches indépendantes, ou des tâches couplées, devant être calculées ensemble et au même rythme. Dans chaque cas, le problème est d'abord étudié théoriquement, puis des heuristiques sont proposées, et leur performances sont comparées.

Mots-clés :
Application de flots de données, optimisation multi-critères, programmes linéaires, plate-formes hétérogènes, heuristiques, complexité, fiabilité, grille de volontaires, pannes transitoires.

Abstract:
This thesis deals with the mapping and the scheduling of workflows. In this context, we consider unreliable platforms, with processors subject to failures.
In a first part, we consider a particular model of streaming applications : the filtering services. In this model, each task impacts the size of its input data by a fixed ratio, increasing or decreasing the size of data. In this context, we aim to obtain the result of any computation as soon as possible (what means minimize the latency), and execute as many computations as possible par time unit(what means minimize the period). We first neglect communication costs. In this model, we study the mono-criterion and the multi-criteria scheduling problems on homogeneous and heterogeneous platforms. Then, many communication models are described, and their impact on scheduling problems of a filtering application is studied. Finally, we consider the scheduling problem of such an application on a chain of processors. The theoretical complexity of any variant of this problem is proved.
This filtering property can model the reliability of processors.The results of some computations are successfully computed, and some other ones are lost. We consider the more frequent failure types : transient failures. Such failure impacts a processor during a very short period of time. Such problem can be caused by a power loss. We consider that transient failures are instantaneous and only influence the current computation. Therefor, we aim efficient and reliable schedules. Three criteria are considered : the period, the reliability and the probability of success of computations. The probability of failures occurrences is supposed constant over time. The complexity of any variant of this problem is proved. Two heuristics are proposed and compared using using simulations.
Even if transient failures are the most common failures in classical grids, some particular type of platform are more concerned by other type of problems. In particular, desktop grids are especially unstable. In desktop grid, users offer idle time of its personal computer. At any time, processors can be turned off. In this context, we want to execute iterative applications : An iterative application is a set of tasks that have to be executed several times. All tasks are executed, then a synchronization occurs, after this synchronization, tasks are executed again, and so on. Two variants of this problem are considered: applications of independent tasks, and applications where all tasks need to be executed at same speed. In both cases, the problem is first theoretically studied, then heuristics are proposed and compared using using simulations.

Keywords:
Streaming applications, multicriteria optimization, linear programs, heterogeneous plateforms, heuristics, complexity results, reliability, desktop grids, transient failures.