On the Complexity of Mapping Pipelined Filtering Services on Heterogeneous Platforms

Anne Benoit, Fanny Dufossé, Yves Robert

On the Complexity of Mapping Pipelined Filtering Services on Heterogeneous Platforms

Anne Benoit, Fanny Dufossé and Yves Robert

LIP, École Normale Supérieure de Lyon, France
{Anne.Benoit|Fanny.Dufosse|Yves.Robert}@ens-lyon.fr

October 10, 2008

LIP Research Report RR-2008-30

Abstract

In this paper, we explore the problem of mapping filtering services on large-scale heterogeneous platforms. Two important optimization criteria should be considered in such a framework. The period, which is the inverse of the throughput, measures the rate at which data sets can enter the system. The latency measures the response time of the system in order to process one single data set entirely. Both criteria are antagonistic. For homogeneous platforms, the complexity of period minimization is already known [1]; we derive an algorithm to solve the latency minimization problem in the general case with service precedence constraints; for independent services we also show that the bi-criteria problem (latency minimization without exceeding a prescribed value for the period) is of polynomial complexity. However, when adding heterogeneity to the platform, we prove that minimizing the period or the latency becomes NP-hard, and that these problems cannot be approximated by any constant factor (unless P=NP). The latter results hold true even for independent services. We provide an integer linear program to solve both problems in the heterogeneous case with independent services.

For period minimization on heterogeneous platforms, we design some efficient polynomial time heuristics and we assess their relative and absolute performance through a set of experiments. For small problem instances, the results are very close to the optimal solution returned by the integer linear program.

Key words: query optimization, web service, filter, workflow, period, latency, complexity results.

1 Introduction

This paper deals with the problem of query optimization over web services [1, 2]. The problem is close to the problem of mapping pipelined workflows onto distributed architectures, but involves several additional difficulties due to the filtering properties of the services.

In a nutshell, pipelined workflows are a popular programming paradigm for streaming applications like video and audio encoding and decoding, DSP applications etc [3, 4, 5]. 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). The goal is to map each node onto some processor so as to optimize some scheduling objective. Since data continually flows 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 [6, 7, 8, 9].

In the query optimization problem, we have a collection of various services that must be applied on a stream of consecutive data sets. As for 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). Also, 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). But in addition, 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 $\sigma_i > 1$. Each service has an elementary cost c_i, which represents the volume of computations required to process a data set of size δ_0. But the volume of computations is proportional to the actual size of the input data, which may have shrunk or expanded by the predecessors of C_i in the mapping. Altogether, the time to execute a data set of size $\sigma \times \delta_0$ when service C_i is mapped onto server S_u of speed s_u is $\sigma \frac{c_i}{s_u}$. Here σ denotes the combined selectivity of all predecessor of C_i in the mapping.

Consider now 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 (fake) 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, leading to a better solution than running both services in parallel. 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 service with small selectivity. 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 1, then the cost of C_2 is $\sigma_1 c_2$ and the cost of C_3 is $\sigma_1\sigma_2 c_3$. If C_i is mapped onto S_i, for $i = 1, 2, 3$, then the period is $\mathcal{P} = \max\left(\frac{c_1}{s_1}, \frac{\sigma_1 c_2}{s_2}, \frac{\sigma_1\sigma_2 c_3}{s_3}\right)$, while the latency is $\mathcal{L} = \frac{c_1}{s_1} + \frac{\sigma_1 c_2}{s_2} + \frac{\sigma_1\sigma_2 c_3}{s_3}$. 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 1 or in Figure 2, then the cost of C_3 becomes $\sigma_1\sigma_2c_3$. With the mapping of Figure 2, the period is $\mathcal{P} = \max\left(\frac{c_1}{s_1}, \frac{c_2}{s_2}, \frac{\sigma_1\sigma_2c_3}{s_3}\right)$, while the latency is $\mathcal{L} = \max\left(\frac{c_1}{s_1}, \frac{c_2}{s_2}\right) + \frac{\sigma_1\sigma_2c_3}{s_3}$. We see from the latter formulas that the model neglects the cost of *joins* when combining two services as predecessors of a third one.

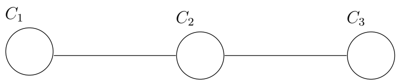

Figure 1: Chaining services.

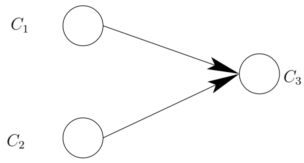

Figure 2: Combining selectivities

All hypotheses and mapping rules are those of Srivastava et al. [1, 2]. Although their papers mainly deal with query optimization over web services (already an increasingly important application with the advent of Web Service Management Systems [10, 11]), the approach applies to general data streams [12] and to database predicate processing [13, 14]. Finally (and quite surprisingly), we note that our framework is quite similar to the problem of scheduling unreliable jobs on parallel machines [15] where service selectivities correspond to job failure probabilities.

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, first with identical servers, and then with different-speed servers.

In this paper, we establish several new and important complexity results. First 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 [1]. 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 problems unless P=NP and present an integer linear program to solve both problems. We also assess the complexity of several particular problem instances.

The rest of this paper is organized as follows. First we formally state the optimization problems that we address in Section 2. Next we detail two little examples aimed at showing the intrinsic combinatorial complexity of the problem (Section 3). Then Section 4 is devoted to problems with identical resources (homogeneous platforms), while Section 5 is the counterpart for different-speed processors (heterogeneous platforms). We provide a set of heuristics and experiments for period minimization in Sections 6 and 7. Finally we give some conclusions and perspectives in Section 8.

2 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 $\mathcal{F} = \{C_1, C_2, \dots, C_n\}$ 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 c_i and its selectivity σ_i.

For the computing resources, we have a set $\mathcal{S} = \{S_1, S_2, \dots, S_p\}$ 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. We always assume that there are more servers available than services (hence $n \le p$), 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 $S_{\mathsf{alloc}(i)}$.

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 $(C_i, S_{\mathsf{alloc}(i)}) \in \mathcal{C}$, 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 period or the latency. $\mathsf{Ancest}_j(G)$ 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 $cost_i(G) = \left(\prod_{C_j \in \mathsf{Ancest}_i(G)} \sigma_j\right) \times \frac{c_i}{s_{\mathsf{alloc}(i)}}$. We note $L_G(C_i)$ 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. We can now formally define the period $\mathcal{P}$ and latency $\mathcal{L}$ of a plan G:

$$\mathcal{P}(G) = \max_{(C_i,S_u) \in \mathcal{C}} cost_i(G) \quad \text{and} \quad \mathcal{L}(G) = \max_{(C_i,S_u) \in \mathcal{C}} L_G(C_i).$$

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 homogeneous platforms while MinLatency-Prec-Het is the problem of minimizing the latency for arbitrary precedence constraints on heterogeneous platforms.

3 Motivating examples

In this section we deal with two little examples. The first one considers independent services and different-speed processors (hence a problem Indep-Het), while the second one involves precedence constraints and identical resources (Prec-Hom).

[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 $\mathcal{E}$.

3.1 An example for the Indep-Het problem

Consider a problem instance with three independent services C_1, C_2 and C_3. Assume that $c_1 = 1$, $c_2 = 4$, $c_3 = 10$, and that $\sigma_1 = \frac{1}{2}$, $\sigma_2 = \sigma_3 = \frac{1}{3}$. Suppose that we have three servers of respective speeds $s_1 = 1$, $s_2 = 2$ and $s_3 = 3$. What is the mapping which minimizes the period? and same question for the latency? We have to decide for an assignment of services to servers, and to build the best plan.

For MinPeriod-Indep-Het (period optimization), we can look for a plan with a period smaller than or equal to 1. In order to obtain an execution time smaller than or equal to 1 for service C_3, we need the selectivity of C_1 and C_2, and either server S_2 or server S_3. Server S_2 is fast enough to render the time of C_3 smaller than 1, so we decide to assign C_3 to S_2. Service C_2 also needs the selectivity of C_1 and a server of speed strictly greater than 1 to obtain an execution time less than 1. Thus, we assign C_1 to S_1 and make it a predecessor of C_2. In turn we assign C_2 to S_3 and make it a predecessor of C_3. We obtain a period of $\min\left(\frac{1}{1}, \frac{1}{2}\frac{4}{3}, \frac{1}{2\times 3}\frac{10}{2}\right) = 1$. It is the optimal solution. In this plan, the latency is equal to $1 + \frac{4}{6} + \frac{10}{12} = \frac{5}{2}$.

For MinLatency-Indep-Het (latency optimization), we have a first bound: $\frac{5}{2}$. Because of its cost, service C_3 needs at least one predecessor. If C_1 is the only predecessor of C_3, we have to assign C_3 to S_3 in order to keep the latency under $\frac{5}{2}$. The fastest computation time that we can then obtain for C_3 is $\frac{1}{2} + \frac{1}{2}\frac{10}{3}$, with C_1 assigned to S_2. In this case, the fastest completion time for C_2 is $\frac{5}{2}$: this is achieved by letting C_2 be a successor of C_1 in parallel with C_3. Suppose now that C_2 is a predecessor of C_3, and that there is an optimal solution in which C_2 is the only predecessor of C_3. Independently of the choice of the servers assigned to C_1 and C_2, if we put C_1 without any predecessor, it will end before C_2. So, we can make it a predecessor of C_3 without increasing its completion time. So, we are looking for a solution in which C_1 and C_2 are predecessors of C_3. There are three possibilities left: (i) C_1 is a predecessor of C_2; (ii) C_2 is a predecessor of C_1; and (iii) C_1 and C_2 have no predecessors. In the first two cases, we compute for each service a cost weighted by the product of the selectivities of its predecessors. Then, we associate the fastest server to the service with the longest weighted cost and so on. We obtain $\frac{5}{2}$ in both cases. For the last case, we know that the real cost of C_1 will have no influence on the latency, hence we assign it to the slowest server S_1. The weighted cost of the remaining services is 4 for C_2 and $\frac{10}{6}$ for C_3. So, we assign S_3 to C_2 and S_2 to C_3. We obtain a latency of $\frac{4}{3} + \frac{1}{2\times 3}\frac{10}{2} = \frac{13}{6}$. We cannot obtain a strictly faster solution if C_2 is not a predecessor of C_3. As a result, $\frac{13}{6}$ is the optimal latency. In this optimal plan for the latency, the period is $\frac{4}{3}$.

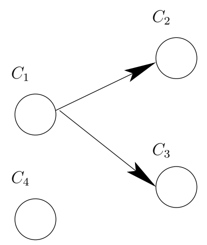

Figure 3: Precedence constraints.

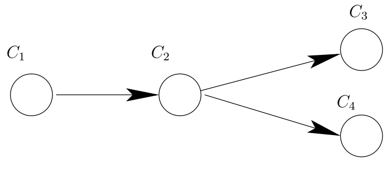

Figure 4: Optimal plan for period.

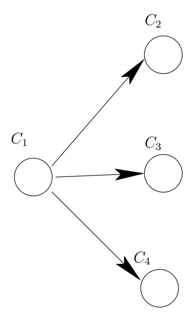

Figure 5: Optimal plan for latency.

3.2 An example for the Prec-Hom problem

Let $\mathcal{A} = (\mathcal{F}, \mathcal{G})$ be the following set of 4 services : $c_1 = c_2 = 1$, $c_3 = c_4 = 4$, $\sigma_1 = \frac{1}{2}$, $\sigma_2 = \frac{4}{5}$, $\sigma_3 = \sigma_4 = 2$ and $\mathcal{G} = \{(C_1, C_2), (C_1, C_3)\}$ (see Figure 3). With this dependence set, we have 3 possible combinaisons for ordering C_1, C_2, C_3, and for each of these orderings, 10 possible graphs when adding C_4. We target a homogeneous platform with four identical servers of speed $s = 1$.

For MinPeriod-Prec-Hom, suppose that we can obtain a period strictly less than 2. C_1 is the only service that can be placed without predecessor, because $c_4 > 2$, and both C_2 and C_3 need C_1 as an ancestor (precedence constraints). C_2 is the only remaining service of cost strictly less than 4. It can be placed with C_1 as unique predecessor. Then we place C_3 and C_4 with predecessors C_1 and C_2. We obtain a period $P = \frac{8}{5}$ (see Figure 4), which is optimal.

Let us study MinLatency-Prec-Hom. With the plan shown in Figure 5, we obtain a latency $L = 1 + \frac{1}{2} \times 4 = 3$. Suppose that we can obtain a latency strictly less than 3. Again, C_1 is the only service that can be placed without any predecessor. As for MinPeriod, C_2 is the only service that can be placed after C_1. But in this case, C_3 and C_4 cannot be placed after C_2, because it would give a latency $L = 1 + \frac{1}{2} \times 1 + \frac{1}{2} \times \frac{4}{5} \times 4 > 3$. Therefore, 3 is the optimal latency for this problem instance.

4 Homogeneous platforms

In this section we investigate the optimization problems with homogeneous resources. Problem MinPeriod-Prec-Hom (minimizing the period with precedence constraints and identical resources) was shown to have polynomial complexity in [1, 2]. We show that 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.

4.1 Latency

We describe here optimal algorithms for MinLatency-Hom, without dependences first, and then for the general case.

Theorem 1. (Independent services) *Algorithm 1 computes the optimal plan for* MinLatency-Indep-Hom *in time $O(n^2)$.*

Data: n independent services with selectivities $\sigma_1, ..., \sigma_p \leq 1$, $\sigma_{p+1}, ..., \sigma_n > 1$, and ordered costs $c_1 \leq \cdots \leq c_p$
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 $C_1 \to C_i, \ldots, C_j \to C_i$ to G;
end

Algorithm 1: Optimal algorithm for MinLatency-Indep-Hom.

Proof. We show that Algorithm 1 verifies the following properties:

- (A) $L_G(C_1) \leq L_G(C_2) \leq \cdots \leq L_G(C_p)$
- (B) $\forall i \leq n$, $L_G(C_i)$ is optimal

Because the latency of any plan G' is the completion time of its last node (a node C_i such that $\forall C_j, L_{G'}(C_i) \geq L_{G'}(C_j)$), property (B) shows that $\mathcal{L}(G)$ is the optimal latency. We prove properties (A) and (B) by induction on i: for every i we prove that $L_G(C_i)$ is optimal and that $L_G(C_1) \leq L_G(C_2) \leq \cdots \leq L_G(C_i)$.

For $i = 1$: C_1 has no predecessor in G, so $L_G(C_1) = c_1$. Suppose that there exists G' such that $L_{G'}(C_1) < L_G(C_1)$. If C_1 has no predecessor in G', then $L_{G'}(C_1) = c_1 = L_G(C_1)$. Otherwise, let C_i be a predecessor of C_1 in G' such that C_i has no predecessor itself. $L_{G'}(C_1) > c_i \geq c_1$. In both cases, we obtain a contradiction with the hypothesis $L_{G'}(C_1) < L_G(C_1)$. So $L_G(C_1)$ is optimal.

Suppose that for a fixed $i \leq p$, $L_G(C_1) \leq L_G(C_2) \leq \cdots \leq L_G(C_{i-1})$ and $\forall j < i$, $L_G(C_j)$ is optimal. Suppose that there exists G' such that $L_{G'}(C_i)$ is optimal. Let C_k be the predecessor of C_i of greatest cost in G'. If $c_k > c_i$, we can choose in G' the same predecessors for C_i than for C_k, thus strictly reducing $L_{G'}(C_i)$. However, $L_{G'}(C_i)$ is optimal. So, we obtain a contradiction and $c_k \leq c_i$. Thus,

$$\begin{aligned}
L_{G'}(C_i) &= L_{G'}(C_k) + \left(\prod_{C_j \in \mathsf{Ancest} L_{G'}(C_i)} \sigma_j\right) c_i & \\
&\geq L_{G'}(C_k) + \left(\prod_{j \leq k} \sigma_j\right) c_i & \text{by definition of } C_k \\
&\geq L_G(C_i) & \text{by construction of } G
\end{aligned}$$

Therefore, since $L_{G'}(C_i)$ is optimal by hypothesis, we have $L_{G'}(C_i) = L_G(C_i)$.
Suppose now that $L_G(C_i) < L_G(C_{i-1})$. 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

$$\begin{aligned}
L_{G''}(C_{i-1}) &= \max_{k \leq j} L_G(C_k) + \prod_{k \leq j} \sigma_k c_{i-1} & \text{by construction of node } C_{i-1} \\
&\leq \max_{k \leq j} L_G(C_k) + \prod_{k \leq j} \sigma_k c_i = L_G(C_i) &
\end{aligned}$$

However, $L_G(C_{i-1})$ is optimal, and so $L_G(C_{i-1}) \leq L_{G''}(C_{i-1}) \leq L_G(C_i)$, which leads to a contradiction. Therefore, $L_G(C_1) \leq L_G(C_2) \leq \cdots \leq L_G(C_i)$.

At this point, we have proved that the placement of all services of selectivity smaller than 1 is optimal, and that $L_G(C_1) \leq L_G(C_2) \leq \cdots \leq L_G(C_p)$. We now proceed with services C_{p+1} to C_n.

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

$$\begin{aligned}
\max_{j \leq k} L_G(C_j) + \prod_{j \leq k} \sigma_j * c_i &= \max_{j=p}^{k} L_G(C_j) + \prod_{j=1}^{k} \sigma_j * c_i \\
&\geq L_G(C_p) + \prod_{j \leq k} \sigma_j * c_i \\
&> L_G(C_p) + \prod_{j \leq p} \sigma_j * c_i
\end{aligned}$$

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 $L_{G'}(C_i)$ 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 $L_{G'}(C_i) \geq L_G(C_i)$, and thus $L_G(C_i)$ is optimal. □

Theorem 2. (General case) *Algorithm 2 computes the optimal plan for* MinLatency-Prec-Hom *in time* $O(n^6)$*.*

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** $C \in S$ **do**
6   **for** $C' \in G$ **do**
7   Compute the set S' minimizing the product of selectivities among services of latency less than $L_G(C')$, and including all predecessors of C in $\mathcal{G}$ (using an algorithm from [2], whose execution time is $O(n^3)$);
8   **end**
9   Let S_C be the set that minimizes the latency of C in G and L_C be this latency;
10   **end**
11   Choose a service C such that $L_C = \min\{L_{C'}, C' \in S\}$;
12   Add to G the node C, and $\forall C' \in S_C$, the edge $C' \rightarrow C$;
13 **end**

Algorithm 2: Optimal algorithm for MinLatency-Prec-Hom.

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 $L_G(C_1) \leq L_G(C_2) \leq \ldots \leq L_G(C_n)$, and G is optimal for $L(C_i)$, $1 \leq i \leq n$. 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, $L_G(C_1) = c_1$. Let G' be a valid plan such that $L_{G'}(C_1)$ is optimal. If C_1 has no predecessors in G', then $L_{G'}(C_1) = L_G(C_1)$. 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 $c_1 \leq c_i$. Therefore, $L_{G'}(C_1) \geq c_i \geq c_1 = L_G(C_1)$. Since $L_{G'}(C_1)$ is optimal, $L_G(C_1) = L_{G'}(C_1)$ and thus $L_G(C_1)$ is optimal.

Suppose that for a fixed $i \leq n$, we have $L_G(C_1) \leq L_G(C_2) \leq \ldots \leq L_G(C_{i-1})$, and $\forall j < i, L_G(C_j)$ is optimal. Let us prove first that $L_G(C_{i-1}) \leq L_G(C_i)$. If C_{i-1} is a predecessor of C_i, then the result is true. Otherwise, and if $L_G(C_{i-1}) > L_G(C_i)$, then C_i would have been chosen at step $i-1$ of the algorithm (line 9) instead of C_{i-1}, which leads to a contradiction. It remains to prove that $L_G(C_i)$ is optimal. Let us consider a valid plan G' such that $L_{G'}(C_i)$ 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 $L_{G'}(C_k)$ 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 $L_{C_k} \leq L_{G'}(C_k)$ (see Property A below). Then $L_G(C_i) \leq L_{C_k} \leq L_{G'}(C_k) \leq L_{G'}(C_i)$.
(ii) If the set of predecessors of C_i in G' is included in $\{C_1, ..., C_{i-1}\}$, then we can prove that $L_{G'}(C_i) \geq L_{C_i} = L_G(C_i)$, where L_{C_i} is the value computed at step i (see Property B below).

In both cases (i) and (ii), since $L_{G'}(C_i)$ is optimal, we have $L_G(C_i) = L_{G'}(C_i)$, thus proving the optimality of $L_G(C_i)$.

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 $L_{G'}(C_k) \geq L_{C_k}$, 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 $j < i$, $L_G(C_j)$ 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 $L_{G'}(C_k) \geq L_{C_k}$.

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

4.2 Bi-criteria problem

Theorem 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* $O(n^6)$.

Data: n services with selectivities $\sigma_1, ..., \sigma_p \leq 1$, $\sigma_{p+1}, ..., \sigma_n > 1$, ordered costs $c_1 \leq \cdots \leq c_p$, and a maximum period K
Result: a plan G optimizing the latency with a period less than K

G is the graph reduced to node C_1;
if $c_1 > K$ **then**
    return false;
end
for $i = 2$ ***to*** n **do**
    for $j = 0$ ***to*** $i - 1$ **do**
        Compute the completion time t_j of C_i in G with predecessors $C_1, ..., C_j$;
    end
    Let $S = \{k | c_i \prod_{1 \leq l \leq k} \sigma_l \leq K\}$;
    if $S = \emptyset$ **then**
        return false;
    end
    Choose j such that $t_j = \min_{k \in S}\{t_k\}$;
    Add the node c_i and the edges $C_1 \to C_i$, $\ldots$, $C_j \to C_i$ to G;
end

Algorithm 3: Optimal algorithm BiCriteria-Indep-Hom.

Proposition 1. *Algorithm 3 computes the optimal latency for a bounded period with independent services (problem* BiCriteria-Indep-Hom*).*

Proof. The proof is similar to that of Theorem 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. □

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 $C \in S$ **do**
    for $C' \in G$ **do**
      Compute the set S' minimizing the product of selectivities among services of latency less than $L_G(C')$, and including all predecessors of C in $\mathcal{G}$ (using an algorithm from [2], whose execution time is $O(n^3)$);
    end
    Let S_C be the set that minimizes the latency of C in G with a period bounded by K, L_C be this latency and P_C be the computation time of C with the set of predecessors S_C;
  end
  if $\{C', C' \in S \ \ \textit{and}\ P_C \leq K\} = \emptyset$ **then**
    return false;
  end
  Choose a service C such that $L_C = \min\{L_{C'}, C' \in S \ \text{ and } P_C \leq K\}$;
  Add to G the node C, and $\forall C' \in S_C$, the edge $C' \to C$;
end

Algorithm 4: Optimal algorithm for BiCriteria-Prec-Hom.

Proposition 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. 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 studied. If there is no possible set for every possible services, then the bound for period can not be obtain. □

5 Heterogeneous platforms

In this section we investigate 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. We provide for MinPeriod-Indep-Het and MinLatency-Indep-Het a formulation in terms of an integer linear program. The integer linear program of MinPeriod-Indep-Het (on small problem instances) will be used to assess the absolute performance of the polynomial heuristics that we derive in Section 7.

5.1 Period

In this section, we show that problem MinPeriod-Indep-Het is NP-complete. The following property was presented in [1] for homogeneous platforms, and we extend it to

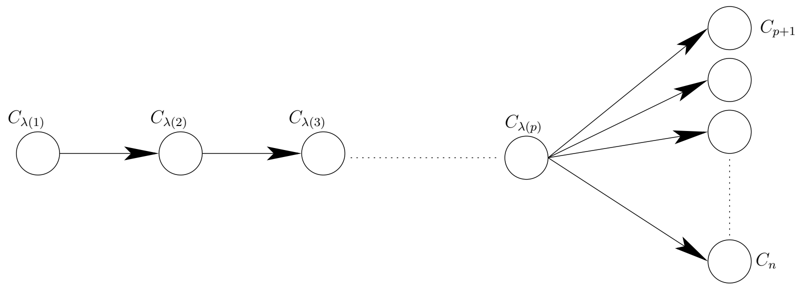

Figure 6: General structure for period minimization.

different-speed servers. We provide an integer linear program and assess the complexity of some particular instances.

Proposition 3. *Let* $(\mathcal{F}, \mathcal{S})$ *be an instance of the problem* MINPERIOD-INDEP-HET. *We suppose* $\sigma_1, \sigma_2, ..., \sigma_p < 1$ *and* $\sigma_{p+1}, ..., \sigma_n \geq 1$*. Then the optimal period is obtained with a plan as in Figure 6.*

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:

$$cost_k(G') = \frac{c_k}{s_u} \times \prod_{C_l \in \mathsf{Ancest}_k(G')} \sigma_l \leq \frac{c_k}{s_u} \times \prod_{C_l \in A'_k(G)} \sigma_l \leq \frac{c_k}{s_u} \times \prod_{C_l \in \mathsf{Ancest}_k(G)} \sigma_l \leq cost_k(G).$$

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 6. □

We point out that only the *structure* of the plan is specified by Proposition 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 6), and to find the optimal assignment of services to servers.

Theorem 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 [16, 17]. 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

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

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. We can suppose 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 1 is equivalent to

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

We build the following instance $\mathcal{I}_2$ of MinPeriod-Het with n services and $p = n$ servers such that $c_i = 2^{A[i]}$, $\sigma_i = 1/2$, $s_i = 2^i$ 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 c_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 previous services, and it is processed by server $S_{\lambda_2(i)}$, matching the cost in Equation 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 is satisfied for all i, hence Equation 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 4. *For any $K > 0$, there exists no K-approximation algorithm for* MinPeriod-Indep-Het*, unless P=NP.*

Proof. In addition, we obtain $cost_i(G) = 2$ for each i. Suppose that there exists an optimal plan G' that is not a chain. According to Proposition 3, it can be transformed step by step into a chain. Suppose that this chain is G. At each step of transformation, we consider a pair (C, C') and we add an edge $C \to C'$. Suppose that at a step, we add the edge $C_i \to C_j$. That means that $\mathsf{Ancest}_j(G') \subsetneq \mathsf{Ancest}_j(G)$. However $cost_j(G) = 2$ and all the selectivities are strictly lower than 1. Then $cost_j(G') > 2$. That contradicts the hypothesis of optimality of G'. This proves that the only optimal plans are chains.

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 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 $1 \leq i \leq n$, $c_i = (2K)^{A[i]-1}$, $\sigma_i = \frac{1}{2K}$, $s_i = (2K)^i$ 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 $P \neq NP$. □

5.1.1 Particular instances

In this section, we study three particular instances of MinPeriod.

Mapping services of selectivity greater than one Let $\mathcal{I}$ be an instance of MinPeriod-Het such that all services have a selectivity greater than 1. We want to know if there exists a plan with a period less than K. For every service C_i, we choose the slowest available server of speed greater than K/c_i. This greedy algorithm is easily seen to be optimal.

The same algorithm holds in the general case, for mapping the subset of services of selectivity greater than 1. We make an hypothesis about the longest ratio cost/speed of those services, and we allocate the slowest possible servers according to this hypothesis. We can then deal with other services. There is a polynomial number of values for the longest ratio cost/speed for services of selectivity greater than 1, i.e., the ratio cost/speed for every service and server.

Case of homogeneous servers The problem MinPeriod-Hom can be solved in polynomial time: see the algorithm in [1]. The structure of the solution is described in Section 5.1, and the optimal placement of the services of selectivity less than one is done by increasing order of costs.

Case of equal selectivities This sub-problem is NP-complete. The proof is the same than for MinPeriod-Het: in the instance $\mathcal{I}_2$ used in the demonstration, the selectivities of all services are equal (to 1/2).

5.1.2 Integer linear program

We present here a linear program to compute the optimal solution of MinPeriod-Het. Let n be the number of services. First, we need to define a few variables:

- For each service C_i and each server S_u, $t_{i,u}$ is a boolean variable equal to 1 if service C_i is assigned to server S_u (and 0 otherwise).
- For each pair of services C_i and C_j, $s_{i,j}$ is a boolean variable equal to 1 if service C_i is an ancestor of C_j (and 0 otherwise).
- M is the logarithm of the optimal period.

We list below the constraints that need to be enforced. First, there are constraints for the matching between services and servers and for the plan:

- Each service is mapped on exactly one server:

$$\forall i, \quad \sum_u t_{i,u} = 1$$

- Each server executes exactly one service:

$$\forall u, \quad \sum_i t_{i,u} = 1$$

- The property "is ancestor of" is transitive: if C_i, C_j, C_k are three services such that $s_{i,j} = 1$ and $s_{j,k} = 1$, then $s_{i,k} = 1$. We write this constraint as:

$$\forall i, j, k, \quad s_{i,j} + s_{j,k} - 1 \leq s_{i,k}$$

- The precedence graph is acyclic:

$$\forall i, s_{i,i} = 0$$

- There remains to express the logarithm of the period of each service and to constrain it by M:

$$\forall i, \quad \log c_i - \sum_u t_{i,u} \log s_u + \sum_k s_{k,i} \log \sigma_k \leq M$$

  In this formula, $\sum_u t_{i,u} \log s_u$ accounts for the speed of the server which processes C_i, and $\sum_k s_{k,i} \log \sigma_k$ adds selectivities of all predecessors of C_i.

Finally, the objective function is to minimize the period M. We have $O(n^2)$ variables, and $O(n^3)$ constraints. All variables are boolean, except M, the logarithm of the period. This integer linear program has been implemented with CPLEX [18], and the code is available in [19]

5.2 Latency

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. We provide an integer linear program and assess the complexity of some particular instances.

Definition 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 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 $c_{i_1}/s_{u_1} < c_{i_2}/s_{u_2}$.*
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 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 $c_i = \max_{C_j \in A_0} c_j$. 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: $L_{G'}(C_\ell) = L_G(C_\ell)$. And, for each service C_k in A_0,

$$\begin{array}{rcl} L_{G'}(C_k) & = & L_{G'}(C_j) + \left(\prod_{C_\ell \in B_i} \sigma_\ell\right) \times c_k \\ & \leq & L_{G'}(C_j) + \left(\prod_{C_\ell \in B_i} \sigma_\ell\right) \times c_i \\ & \leq & L_{G'}(C_i) = L_G(C_i) \end{array}$$

Therefore, $\forall C_k \in A_0,\ L_{G'}(C_k) \leq L_G(C_i)$. Since for $C_k \notin A_0, L_{G'}(C_k) \leq L_G(C_k)$, 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 $L_{G_i}(C_{i'}) = \max_k L_{G_i}(C_k)$. 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 $L_{G_i}(C_{i'})$, and let C_ℓ be the predecessor in G of C_j of maximal finish time $L_G(C_\ell)$. 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 $L_{G'}(C_k) = L_G(C_k)$. Let C_k be a node of G_i. We have:

$$\begin{array}{rcl} L_{G'}(C_k) & = & L_{G'}(C_\ell) + \left(\prod_{C_m \in \mathsf{Ancest}_j(G')} \sigma_m\right) \times L_{G_i}(C_k) \\ & \leq & L_G(C_\ell) + \left(\prod_{C_m \in \mathsf{Ancest}_j(G)} \sigma_m\right) \times L_{G_i}(C_{i'}) \\ & \leq & L(G) \end{array}$$

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 $c_i/s_u > c_j/s_v$. 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 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 1. *Let $C_1, ..., C_n, S_1, ..., S_n$ be an instance of* MinLatency-Het *such that for all i, c_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 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 $\frac{c_{i_1}}{s_{u_1}} = \frac{c_{i_2}}{s_{u_2}}$.*

Proof. Let G be a plan verifying Proposition 5. Suppose that there exists a distance to leaves d such that the nodes at this distance do not respect Property 2 of Lemma 1. Let A be the set of nodes (C_i, S_u) of maximal ratio $\frac{c_i}{s_u} = c$ with $d_i(G) = d$ and A' be the set of other nodes at distance d. Let c' be the maximal ratio $\frac{c_j}{s_v}$ of nodes $(C_j, S_v) \in A'$. Since $c' < c$ and c, c' are integer power of 2, we have $c' \leq \frac{c}{2}$.

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 $\frac{c_k}{s_v}$. 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

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

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 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 less pairs of nodes that does not correspond to the property of Lemma 1 than in G. In addition, G' respect properties of Proposition 5. By induction, we can obtain a graph as described in Lemma 1. □

Theorem 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 completness, we use a reduction from RN3DM. Consider the following general instance $\mathcal{I}_1$ of RN3DM: given an integer vector $A = (A[1], \dots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \dots, n\}$ such that

$$\forall 1 \leq i \leq n, \quad \lambda_1(i) + \lambda_2(i) = A[i] \tag{3}$$

We can suppose that $\sum_{i=1}^{n} A[i] = n(n+1)$. We build the following instance $\mathcal{I}_2$ of MinLatency-Het such that:

- $c_i = 2^{A[i] \times n + (i-1)}$
- $\sigma_i = \left(\frac{1}{2}\right)^n$
- $s_i = 2^{n \times (i+1)}$
- $K = 2^n - 1$

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

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

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 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 $\frac{c_{i_1}}{s_{u_1}} = 2^{a_1}$ and $\frac{c_{i_2}}{s_{u_2}} = 2^{a_2}$. 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 $\frac{c_{i_1}}{s_{u_1}} \neq \frac{c_{i_2}}{s_{u_2}}$. The ratios cost/speed are all different and G verifies properties of Lemma 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. $\square$

Proposition 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 compute a K-approximation of this problem. We use RN3DM, a special instance of 3-dimensional matching. Let $\mathcal{I}_1$ be an instance of RN3DM: given an integer vector $A = (A[1], \dots, A[n])$ of size $n \geq 2$, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \dots, n\}$ such that

$$\forall 1 \leq i \leq n, \quad \lambda_1(i) + \lambda_2(i) = A[i] \tag{4}$$

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 has no solution.

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

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

We prove in demonstration of NP-completness of this problem 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 greatest or equal to $(2K)^{n^2}$, that means that the latency obtain is $L \geq (2K)^{n^2}$. If there exists an algorithm that compute 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 solutions, 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: $P \neq NP$. This concludes the proof. □

5.2.1 Particular instances

In this section, we study four particular instances of MinLatency-Het.

MinLatency on a chain Let $C_1, ..., C_n, S_1, ..., S_n$ be an instance of MinLatency-Het. The problem studied here is to compute the optimal latency when we impose that the plan is a linear chain. This problem is NP-complete.

Indeed, consider the decision problems associated to this problem: given an instance of the problem with n services and n servers, and a bound K, is there a matching 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 the same problem as for the completeness of MinPeriod-Het: RN3DM. Consider the following general instance $\mathcal{I}_1$ of RN3DM: given an integer vector $A = (A[1], \dots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \dots, n\}$ such that

$$\forall 1 \leq i \leq n, \quad \lambda_1(i) + \lambda_2(i) = A[i] \tag{5}$$

We build the following instance $\mathcal{I}_2$ of MinLatency-Het on a chain with n services and n servers such that $c_i = 2^{A[i]}$, $\sigma_i = 1/2$, $s_i = 2^i$ and $K = 2n$. The proof is based on the fact that for all $u_1, u_2, \ldots, u_n$, we have

$$\frac{2^{u_1} + 2^{u_2} + \cdots + 2^{u_n}}{n} \geq 2^{\frac{u_1+u_2+\cdots+u_n}{n}} \tag{6}$$

because of the convexity of the power function, and with equality if and only if all the u_i are equal. Now we show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ has a solution. Let λ_1, λ_2 be a solution of $\mathcal{I}_1$. We assign service C_i on server $S_{\lambda_1(i)}$ at position $\lambda_2(i)$. We obtain a computing time of 2 for every service and a latency of $2n$. This is a solution of $\mathcal{I}_2$.

Reciprocally, if we have a solution to $\mathcal{I}_2$ λ_1, λ_2, we have

$$\sum_i 2^{A[i]-\lambda_1(i)-\lambda_2(i)+1} = 2n$$

That is the lower bound of the latency on this instance, according to the equation (6). That means that we have $\forall i, A[i] - \lambda_1(i) - \lambda_2(i) = 0$. So, λ_1, λ_2 is a solution of $\mathcal{I}_1$. This completes the proof of NP-completeness.

Services of same cost Let $C_1, ..., C_n$, $S_1, ..., S_n$ be an instance of MinLatency-Het with for all i, $c_i = c$. We suppose $\sigma_1 \leq \cdots \leq \sigma_n$ and $s_1 \geq \cdots \geq s_n$. We prove that an optimal plan is obtained with the mapping $(C_1, S_1), ..., (C_n, S_n)$. Let G be the graph produced by Algorithm 1 with this mapping. Let r be a permutation of $\{1, ..., n\}$, and G' a plan with the mapping $(C_{r(1)}, S_1), ..., (C_{r(n)}, S_n)$. Let G'' the graph obtained by Algorithm 1 with the latter mapping.

We prove by induction on i that

- $\forall i, t_{r(i)}(G') \geq t_{r(i)}(G)$ and
- $t_{r(1)}(G) = t_{r(1)}(G')$.

Indeed, suppose that for all $j < i$, $t_{r(j)}(G') \geq t_{r(j)}(G)$.

$$\begin{array}{rcl} t_{r(i)}(G') & \geq & t_{r(i)}(G'') \\ & \geq & \max_{k<r(i)}\{t_k(G'') + \prod_{k<r(i)} \sigma_k c_{r(i)}\} \\ & \geq & \max_{k<r(i)}\{t_k(G) + \prod_{k<r(i)} \sigma_k c_{r(i)}\} \\ & \geq & t_{r(i)}(G) \end{array}$$

When the optimal plan is a star Let $C_1, ..., C_{n+1}$, $S_1, ..., S_{n+1}$ be an instance of MinLatency-Het such that $\sigma_1, ..., \sigma_n < 1$, $\sigma_{n+1} \geq 1$. We assume that $c_1, ..., c_n$ are close enough so that the optimal plan is like in Figure 7.

We have to allocate servers to services and to choose the predecessors of C_{n+1} in order to obtain a latency $L \leq K$ for a certain K (in an outer procedure, we will perform a binary search to derive the optimal value of K). We suppose that we know the server S allocated to C_{n+1} and its combined selectivity in an optimal graph. Let $c' = c_{n+1}/s$, $K' = \max_{(C_i,S_j)\in V'} c_i/s_j$ where V' the set of predecessors of C_{n+1} and $\Sigma = (K - K')/c'$. We associate to this problem a bipartite weighted graph $G = (A, B, V)$ with:

- A is the set of services
- B is the set of servers

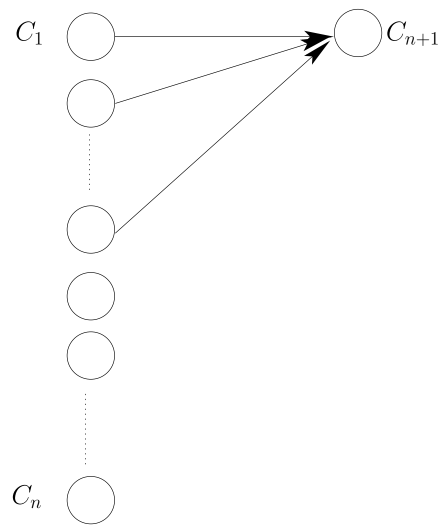

Figure 7: When the optimal plan is a star graph.

- $(C_i, S_j) \in V$ if $c_i/s_j \leq K$
- If $c_i/s_j \leq K'$, then $w(C_i, S_j) = -\ln(\sigma_i)$, and otherwise $w(C_i, S_j) = 0$.

We can compute in polynomial time a perfect matching of maximal weight in this graph. If the associated weight is greater than $\ln \Sigma$, then the associated allocation and plan has a latency $L \leq K$. We can execute this algorithm on all servers that could be allocated to C_{n+1} and on the value of c_i/s_j for all couples (C_i, S_j). So this case is polynomial.

When the optimal plan is a bipartite graph Let $C_1, ..., C_n, S_1, ..., S_n$ be an instance of MinLatency-Het. We suppose in this case that we have n services with $\sigma_1, ..., \sigma_p < 1$ and $\sigma_{p+1}, ..., \sigma_n \geq 1$. We assume that $c_1, ..., c_n$ are close enough so that the optimal plan is like in Figure 8.

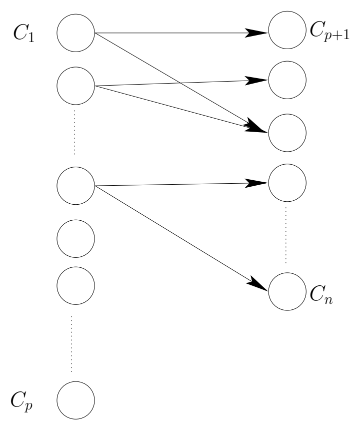

Figure 8: When the optimal plan is a bipartite graph.

In this case, we make an hypothesis on $c' = \max_{p<i\leq n} c_i/s_{\theta(j)}$, with θ the permutation corresponding to the allocation of servers. Then we allocate each service C_{p+i} to the slowest server S possible such that $c_{p+i}/s \leq c'$. We can now use the same algorithm as

for star graphs with the remaining servers and services. We apply this algorithm on each value c_{p+i}/s_i for c'. Again, this case is polynomial

5.2.2 Integer linear program

We present here a linear program to compute the optimal solution of MinLatency-Het. We denote by $\mathcal{C}$ the set of services and by $\mathcal{S}$ the set of servers. First, we need to define a few variables:

- For each service C_i, for each server S_u, and for any subset of services e, $z(i,u,e)$ is a boolean variable equal to 1 if and only if the service C_i is associated to the server S_u and its set of predecessors is $e \subset \mathcal{C}$.
- For each service C_i, the rational variable $t(i)$ is the completion time of C_i.
- The rational variable M is the optimal latency.

We list below the constraints that need to be enforced:

- For each server, there is exactly one service with exactly one set of predecessors:

$$\forall u \in \mathcal{S}, \quad \sum_{i \in \mathcal{C}} \sum_{e \subset \mathcal{C}} z(i,u,e) = 1$$

- Each service has exactly one set of predecessors and is mapped on exactly one server:

$$\forall i \in \mathcal{C}, \quad \sum_{u \in \mathcal{S}} \sum_{e \subset \mathcal{C}} z(i,u,e) = 1$$

- The property "is ancestor of" is transitive:

$$\forall i,i' \in \mathcal{C}, \forall u,u' \in \mathcal{S}, \forall e,e' \subset \mathcal{C}, e \not\subseteq e', i \in e', \quad z(i,u,e) + z(i',u',e') \leq 1$$

- The graph of precedence is acyclic:

$$\forall u \in \mathcal{S}, \forall e \subset \mathcal{C}, \forall i \in e, \ z(i,u,e) = 0$$

- There remains to express the latency of each server and to constrain it by M. First for the case where C_i has some predecessors we write:

$$\forall i \in \mathcal{C}, \forall e \subset \mathcal{C}, \forall k \in e, \quad t(i) \geq \sum_{u \in \mathcal{S}} z(i,u,e) \left(\frac{c_i}{s_u} * \prod_{C_j \in e} \sigma_j + t(k) \right)$$

  But the subset of predecessors can be empty:

$$\forall i \in \mathcal{C}, \quad t(i) \geq \sum_{u} z(i,u,e) \frac{c_i}{s_u} * \prod_{C_j \in e} \sigma_j$$

  Then we bound the value of $t(i)$:

$$\forall i \in \mathcal{C}, \quad t(i) \leq M$$

Finally, the objective function is to minimize the latency M.

We have $O(n^2 * 2^n)$ variables, and $O(n^4 * 2^{2n})$ constraints. All variables are boolean, except the latency M, and the completion times $t(i)$ which are rational. We see that the size of this program is exponential, and it cannot be used in practice, even for small instances.

6 Heuristics

We know that MinPeriod-Het and MinLatency-Het are both NP-complete, but we only propose polynomial heuristics for MinPeriod-Het: in the experiments of Section 7, the absolute performance of these heuristics will be assessed through a comparison with the (optimal) solution returned by the integer linear program of Section 5.1.2. We do not produce heuristics nor experiments for MinLatency-Het, because the integer linear program of Section 5.2.2 is unusable (with $O(2^n)$ variables) and it is untractable for the CPLEX optimization software.

Recall that n is the number of services. The following heuristics are working for instances $C_1, ..., C_n, S_1, ..., S_n$ such that the selectivity of each service is smaller than or equal to 1. The code for all heuristics, implemented in C, is available on the web [19].

Notice that services with selectivity greater than 1 can always be assigned optimally. The idea is to set a bound K for the period, and to assign the slowest possible server to the latter services, in decreasing order of their cost. Then we run the heuristics to assign the services whose selectivity is smaller than 1 (and decrease or increase K according to the result). We can bound the number of iterations in the binary search to be polynomial. Intuitively, the proof goes as follows: we encode all parameters as rational numbers of the form $\frac{\alpha_r}{\beta_r}$, and we bound the number of possible values for the period as a multiple of the least commun multiple of all the integers α_r and β_r. The logarithm of this latter number is polynomial in the problem size, hence the number of iterations of the binary search is polynomial too[3]. Finally, we point out that in practice we expect only a very small number of iterations to be necessary to reach a reasonable precision.

sigma-inc In this first heuristic, we place services on a chain in increasing order of σ. Then, we compute for each service, its cost weighted by the product of the selectivities of its predecessors, and we associate the fastest server to the service with maximum weighted cost, and so on. This heuristic is optimal when all the service costs are equal.

In the next three heuristics, we first allocate servers to services according to some rules. Then, we have for each service its cost weighted by the inverse of the speed of its associated server, and the problem is similar to the homogeneous case. Indeed, we just need to decide how to arrange services. However, we know that this problem can be solved easily in the homogeneous case, since all selectivities are smaller than or equal to 1: we place services on a linear chain, sorted by increasing order of (weighted) costs, regardless of their selectivities.

short service/fast server We associate the service with smallest cost to the server with fastest speed. The idea of this heuristic is to process first services as fast as possible so that their selectivities will help reduce the expected larger cost/speed ratio of the following ones.

long service/fast server We associate the service with largest cost to the server with fastest speed. This heuristic is the opposite of the previous one. It is optimal if all the selectivities are equal to 1. We foresee that it will also give good results for selectivities close to 1.

opt-homo This heuristic is in part randomized. We randomly associate services to servers, and then we use the same procedure (assigning by increasing order of weighted cost) to create a linear chain of services.

[3] The interested reader will find a fully detailed proof for a very similar mapping problem in [20].

greedy min This heuristic simply consists of successively running the previous four heuristics on the problem instance, and returning as a result the best of the four solutions.

random This last heuristic is fully random: we randomly associate services and servers, and we randomly place these pairs on a linear chain.

7 Experiments

Several experiments have been conducted for MinPeriod-Het in order to assess the performance of the heuristics described in Section 6.

We have generated a set of random applications and platforms with $n = 1$ to 100 services and servers. For each value of n, we have randomly generated 300 instances of applications and platforms with similar parameters. Each value of the period in the following plots is an average of 300 results.

We report five main sets of experiments. For each of them, we vary some key parameters to assess the impact of these parameters on the performance of the heuristics. In the first experiment, the service costs and server speeds were randomly chosen as integers between 1 and 100. The selectivities were randomly generated between $\sigma = 0.01$ to 1. In the second and third experiments, the parameters are the same except for the selectivities: in the second experiment, selectivities are randomly chosen between $\sigma = 0.01$ to 0.5 (smaller values), while in the third one they are chosen between $\sigma = 0.51$ to 1 (larger values). In the fourth and fifth experiments, the costs and selectivities are chosen as in the first experiment, but the server speeds are randomly chosen between 1 and 5 for the fourth experiment (large heterogeneity), and between 6 and 10 for the fifth experiment (reduced heterogeneity).

For each experiment we report two sets of results. Figures on the left are for a small number of services and include the optimal solution returned by the integer linear program in addition to the heuristics. Figures on the right are for a large number of services and only compare the heuristics. Indeed, the integer linear program requires a prohibitive execution time, or even fails, as soon as we have 30 services and servers.

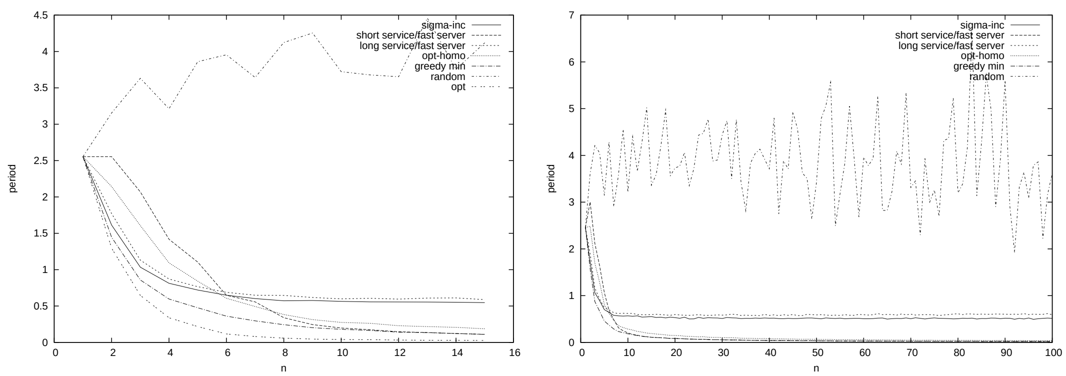

Figure 9: Experiment 1: general experiment.

In the first experiment, we notice that the performance of two heuristics, *sigma-inc* and *long service/fast server*, decreases with the size of n. The two curves are very similar, and they tend towards a constant. These heuristics lead to good results for n small. The heuristic *short service/fast server* obtains the best results for large n, but it is the worst heuristic for small values of n. The heuristic *opt-homo* has correct results for small values of n, and its average period is around twice the average period of the heuristic *short service/fast server* for large values of n. In this experiment, the heuristic *greedy-min* always is very close to the optimal.

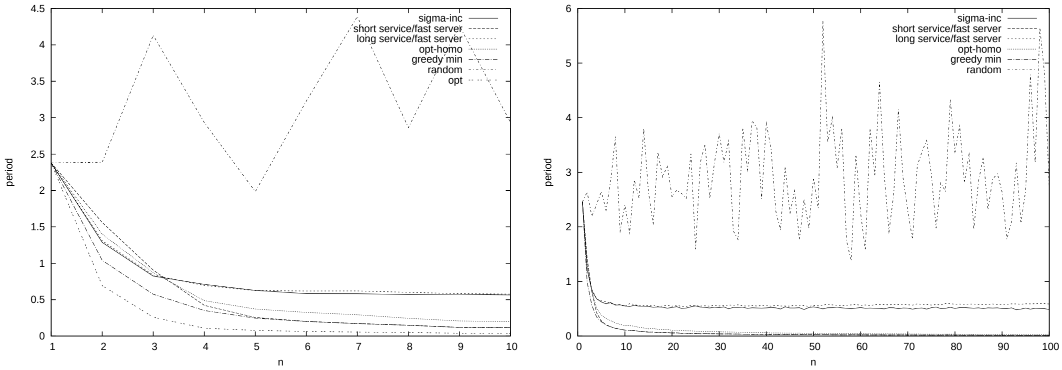

Figure 10: Experiment 2: with small selectivities.

In the second experiment, the performance of the heuristic *short service/fast server* is better than in the first experiment for small values of n. It is the worst heuristic only for $n \leq 3$ while it was even the worst for $n = 6$ in the first experiment. The heuristic *greedy-min* is relatively close to the optimal in this experiment. We might have expected *short service/fast server* to obtain better performances here because selectivities are small, but it turned out not to be the case.

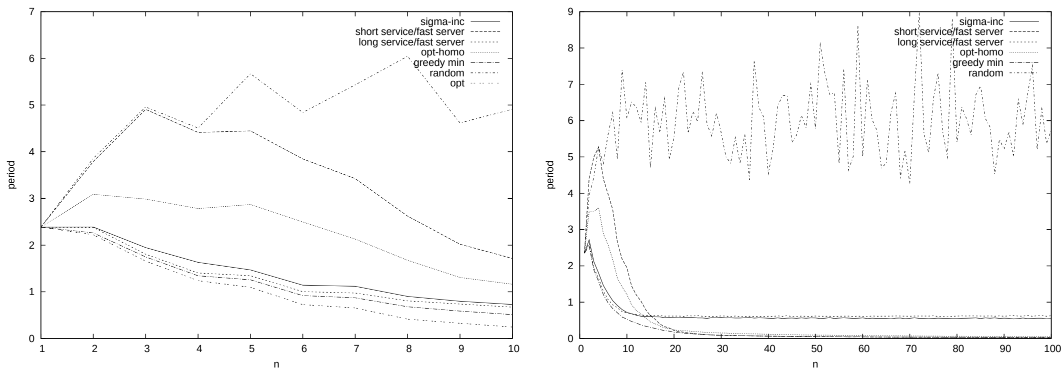

Figure 11: Experiment 3: with big selectivities.

In the third experiment, we expect better results for *long service/fast server* and worse results for *short service/fast server*, since selectivities are closer to 1. This is true for small values of n, but the results for large values of n are similar as before. The heuristic *short service/fast server* is the best when $n > 20$. Altogether, the combination of *long service/fast server* and *sigma-inc* allows *greedy-min* to be very close to the optimal for all the values of n tested.

The fourth experiment is very similar to the first one. We expect similar results with a certain ratio between both experiments. The only difference is the number of cases of equality between server speeds over the instances generated by the two experiments. In practice, the curves of the fourth experiment tend more slowly to constants. The second

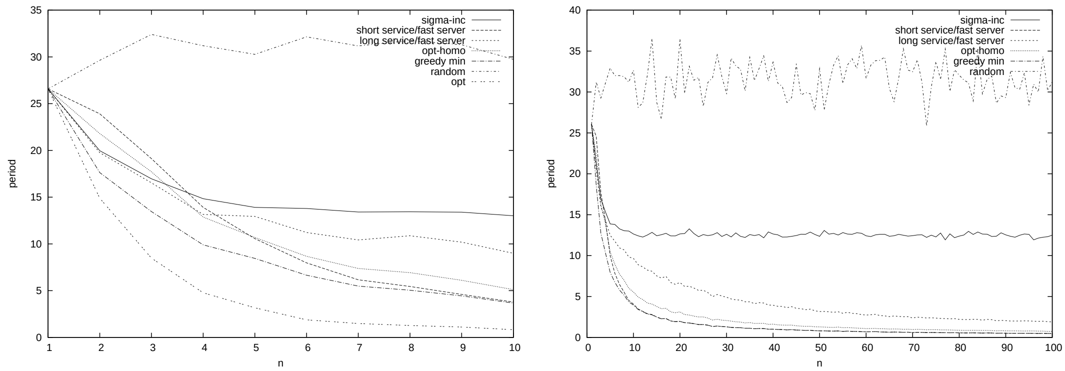

Figure 12: Experiment 4: with high heterogeneity.

difference is the limit of the curves of the heuristics *sigma-inc* and *long service/fast server*. The limit of *sigma-inc* is very high (around 12), but in this experiment, the limit of *long service/fast server* is relatively good (around 2). For this experiment, the heuristics are relatively far from the optimal.

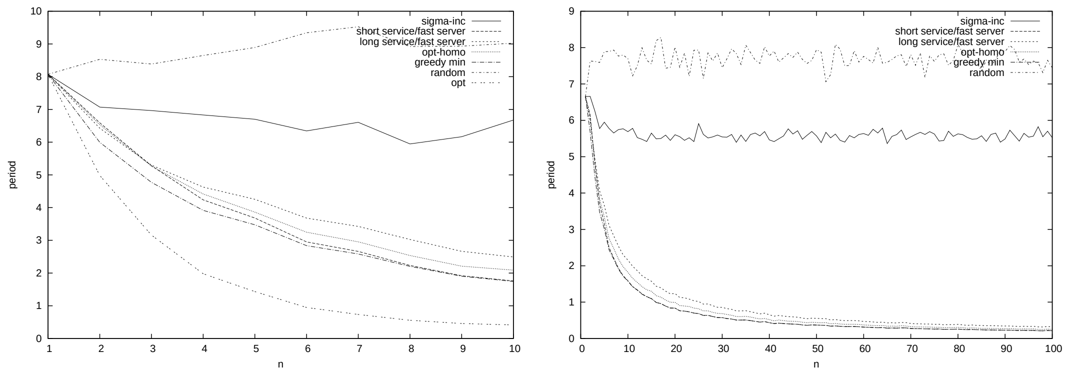

Figure 13: Experiment 5: with small heterogeneity.

We obtain very similar results in the last experiment: it is the only experiment in which the performance of *long service/fast server* is similar to those of *short service/fast server* and *opt-homo*. In this experiment, server speeds are close. It is then logical that the choice of the mapping service/server has a small influence on the result. The heuristic *sigma-inc* has very bad results in this experiment. The instances generated here are close to the homogeneous case. However, the curves generated are somewhat far from the optimal

Figure 14 compares the computing times of the heuristics and of the linear program, according to the size of n. As expected, it takes a long time to solve the linear program (of exponential complexity), while all heuristics always take around 0.001 seconds. For small values of n ($n < 3$), it can seem surprising that the linear program is faster than the heuristics. This artefact can be explained for $n = 1$ by the fact that running the five heuristics implies computing five times the same division (service cost divided by server speed), while the linear program just performs a single addition in this case.

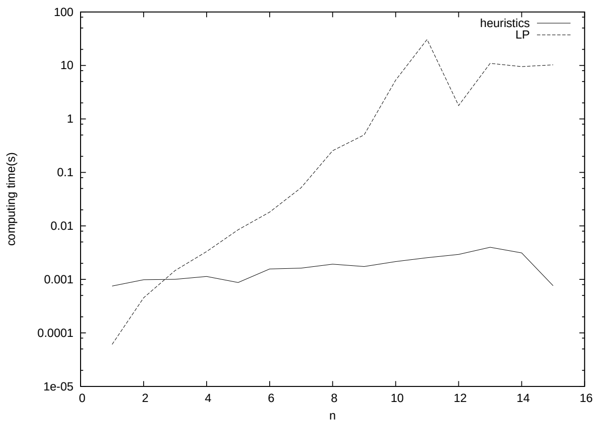

Figure 14: Computing times

8 Conclusion

In this paper, we have considered the important problem of mapping filtering service applications onto computational platforms. Our main focus was to give an insight of the combinatorial nature of the problem, and to assess the impact of using heterogeneous resources on the problem complexity. We considered the two major objective functions, minimizing the period and minimizing the latency, and also studied bi-criteria optimization problems. Several instances of the problem have been shown NP-complete, while others can be solved with complex polynomial algorithms, such as the optimal algorithm for MinLatency-Prec-Hom. We believe that this exhaustive study will provide a solid theoretical foundation for the study of single criterion or bi-criteria mappings.

For Indep-Het, all problems (period, latency, and hence bi-criteria) are NP-hard. We provide an integer linear program and many heuristics for MinPeriod-Indep-Het, and experiments show that in many cases our heuristics are close to the optimal solution returned by the linear program. We also have derived an integer linear program for MinLatency-Indep-Het, but its exponential number of variables renders it untractable, even for small problem instances.

As future work, we intend to design heuristics for the general problems MinPeriod-Prec-Het and MinLatency-Prec-Het, and to derive lower bounds so as to assess their performance. Also, extending ideas of task graph replication algorithms (such as those in [21]) to the framework of pipelined workflows with filtering services looks a promising direction to further explore.

Acknowledgment

This work was supported in part by the ANR StochaGrid project.

References

[1] U. Srivastava, K. Munagala, J. Widom, and R. Motwani, "Query optimization over web services," in *VLDB '06: Proceedings of the 32nd Int. Conference on Very Large Data Bases*. VLDB Endowment, 2006, pp. 355–366.

[2] U. Srivastava, K. Munagala, and J. Burge, "Ordering pipelined query operators with precedence constraints," Stanford University, Research Report 2005-40, November 2005.

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

[4] K. Taura and A. A. Chien, "A heuristic algorithm for mapping communicating tasks on heterogeneous resources," in *Heterogeneous Computing Workshop.* IEEE Computer Society Press, 2000, pp. 102–115.

[5] Q. Wu and Y. Gu, "Supporting distributed application workflows in heterogeneous computing environments," in *14th International Conference on Parallel and Distributed Systems (ICPADS).* IEEE Computer Society Press, 2008.

[6] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz, "An approach for optimizing latency under throughput constraints for application workflows on clusters," Ohio State University, Columbus, OH, Research Report OSU-CISRC-1/07-TR03, Jan. 2007, available at ftp://ftp.cse.ohio-state.edu/pub/tech-report/2007.ShortversionappearsinEuroPar'2008.

[7] ——, "Optimizing latency and throughput of application workflows on clusters," Ohio State University, Columbus, OH, Research Report OSU-CISRC-4/08-TR17, Apr. 2008, available at ftp://ftp.cse.ohio-state.edu/pub/tech-report/2007.

[8] A. Benoit and Y. Robert, "Mapping pipeline skeletons onto heterogeneous platforms," *J. Parallel Distributed Computing*, vol. 68, no. 6, pp. 790–808, 2008.

[9] Q. Wu, J. Gao, M. Zhu, N. Rao, J. Huang, and S. Iyengar, "On optimal resource utilization for distributed remote visualization," *IEEE Trans. Computers*, vol. 57, no. 1, pp. 55–68, 2008.

[10] D. Florescu, A. Grunhagen, and D. Kossmann, "Xl: A platform for web services," in *CIDR 2003, First Biennial Conference on Innovative Data Systems Research*, 2003, on-line proceedings at http://www-db.cs.wisc.edu/cidr/program/p8.pdf.

[11] M. Ouzzani and A. Bouguettaya, "Query processing and optimization on the web," *Distributed and Parallel Databases*, vol. 15, no. 3, pp. 187–218, 2004.

[12] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom, "Adaptive ordering of pipelined stream filters," in *SIGMOD'04: Proceedings of the 2004 ACM SIGMOD Int. Conf. on Management of Data.* ACM Press, 2004, pp. 407–418.

[13] S. Chaudhuri and K. Shim, "Optimization of queries with user-defined predicates," *ACM Trans. Database Systems*, vol. 24, no. 2, pp. 177–228, 1999.

[14] J. M. Hellerstein, "Predicate migration: Optimizing queries with expensive predicates," in *In Proc. of the ACM SIGMOD Conf. on Management of Data*, 1993, pp. 267–276.

[15] M. P. Alessandro Agnetis, Paolo Detti and M. S. Sodhi, "Sequencing unreliable jobs on parallel machines," *Journal on Scheduling*, 2008, available on-lien at http://www.springerlink.com/content/c571u1221560j432.

[16] W. Yu, "The two-machine flow shop problem with delays and the one-machine total tardiness problem," Ph.D. dissertation, Technishe Universiteit Eidhoven, Jun. 1996.

[17] W. Yu, H. Hoogeveen, and J. K. Lenstra, "Minimizing makespan in a two-machine flow shop with delays and unit-time operations is NP-hard," *J. Scheduling*, vol. 7, no. 5, pp. 333–348, 2004.

[18] CPLEX, "ILOG CPLEX: High-performance software for mathematical programming and optimization," http://www.ilog.com/products/cplex/.

[19] F. Dufossé, "Source Code for the Heuristics," http://graal.ens-lyon.fr/~fdufosse/filters/.

[20] L. Marchal, V. Rehn, Y. Robert, and F. Vivien, "Scheduling and data redistribution strategies on star platforms," LIP, ENS Lyon, France, Research Report 2006-23, Jun. 2006.

[21] I. Ahmad and Y.-K. Kwok, "On exploiting task duplication in parallel program scheduling," *IEEE Trans. Parallel Distributed Systems*, vol. 9, no. 9, pp. 872–892, 1998.