Mapping filter services on heterogeneous platforms

Anne Benoit, Fanny Dufossé, Yves Robert

Mapping Filter 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

June 2008

LIP Research Report RR-2008-19

Abstract

In this paper, we explore the problem of mapping filtering web 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 [14]; we derive an algorithm to solve the latency minimization problem, and we provide a bi-criteria algorithm which minimizes latency without exceeding a prescribed value for the period. However, when adding heterogeneity to the platform, we prove that minimizing the period or the latency becomes NP-hard. We provide an integer linear program to solve both problems in the heterogeneous case.

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: web services, filters, scheduling, mapping, period, latency, complexity results, heuristics.

1 Introduction

This paper deals with the problem of mapping web services on large-scale heterogeneous platforms. The main motivation originates from Select-Project-Join query optimization over Web services [14, 12, 13]. We start with an informal and intuitive description of the problem. We refer to Section 2 for a detailed presentation of the framework, and to Section 3 for motivations and a survey of related work.

We may think of the target application as a set of various services that must be applied on a stream of consecutive data sets. We can view each service C_i as a "filter" which operates in pipelined fashion. Consecutive data sets are fed in the service, which processes each data set 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 shrinked (hence the name 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. The volume of computations is proportional to the data size. Each service will be mapped onto a server. If server S_u has speed s_u, then the time to execute a data set of size $\sigma \times \delta_0$ when service C_i is mapped onto server S_u is $\sigma \frac{c_i}{s_u}$.

We assume that the services are independent, which means that they can be applied in any order on each data set (but each data set must be processed by all services). A naive solution is to apply all services in parallel. A better solution may be to chain the execution of some services. Indeed, assume that we chain the execution of two services C_j and C_i, 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_j will decrease in proportion to the volume reduction. 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.

Altogether, we will 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), with the goal of minimizing some important objective function. Such an objective function may be the period (the maximum time for a server to process a data set) or the latency (the time needed for a data set to proceed through all servers).

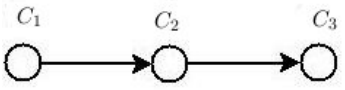

Figure 1: Chaining services.

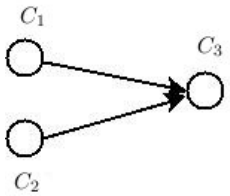

Figure 2: Combining selectivities

We point out 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

$$\mathcal{L} = \frac{c_1}{s_1} + \frac{\sigma_1 c_2}{s_2} + \frac{\sigma_1 \sigma_2 c_3}{s_3}.$$

We point out 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 $\sigma_1 \sigma_2 c_3$. With the mapping of Figure 2, the period is

$$\mathcal{P} = \max\left(\frac{c_1}{s_1}, \frac{c_2}{s_2}, \frac{\sigma_1 \sigma_2 c_3}{s_3}\right),$$

while the latency is

$$\mathcal{L} = \max\left(\frac{c_1}{s_1}, \frac{c_2}{s_2}\right) + \frac{\sigma_1 \sigma_2 c_3}{s_3}.$$

We see from the latter formulas that the model neglects the cost of *joins* when combining two services as predecessors of a third one.

Let us work out a little example in full details. Consider a problem instance with three 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 also to build the mapping graph (also called a *plan*).

For the optimization of the period, 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 the optimization of the latency, 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 obtain in this case 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 that we can obtain 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 $\frac{4}{3}$.

This little example gives a hint on the very combinatorial nature of the problem.

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 different-speed servers.

In this paper, we establish several important complexity results. We prove the NP-completeness of the period minimization problem on a heterogeneous platform. The same problem on homogeneous platforms had been shown to have polynomial complexity in [14]. We introduce a polynomial time algorithm for the latency minimization problem on a homogeneous platform. The complexity of this problem for heterogeneous platforms is left open. For period and latency on heterogeneous platforms, we present two integer linear programs. We also design a polynomial time algorithm for a bi-criteria optimization problem in the homogeneous case. Finally, we design some efficient polynomial time heuristics for period minimization on heterogeneous platforms (the problem which we have shown to be NP-hard), 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.

The rest of this paper is organized as follows. First we formally state the optimization problems that we address in Section 2. Next we give an overview of related work in Section 3. Then Section 4 is devoted to the minimization of the period, while Section 5 is the counterpart for the latency. Section 6 deals with bi-criteria (period/latency) optimization. We provide a set of heuristics and experiments for period minimization in Sections 7 and 8. Finally we give some conclusions and perspectives in Section 9.

2 Framework

As stated above, a web service C_i is characterized by its cost c_i and its selectivity σ_i, while a server S_u is characterized by its speed s_u.

We always assume that there are more servers available than services, 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 $\mathcal{C}$ are couples $(C_i, S_{\mathsf{alloc}(i)})$ and thus define the allocation function. Then we add an arc $e = (C_i, C_j)$ in $\mathcal{E}$ if C_i precedes C_j in the execution. The graph G is called a plan.

Given a plan G, the execution time of a service C_i is

$$cost_i(G) = \prod_{C_j \in \mathsf{Ancest}_i(G)} \sigma_j \times \frac{c_i}{s_{\mathsf{alloc}(i)}}$$

where $\mathsf{Ancest}_i(G)$ denotes the set of all ancestors[1] of C_i in G. We note $t_i(G)$ the completion

[1]The ancestors of a service are the services preceding it, and the predecessors of their predecessors, and

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:

$$\mathcal{P}(G) = \max_{(C_i,S_u)\in\mathcal{C}} cost_i(G)$$

$$\mathcal{L}(G) = \max_{(C_i,S_u)\in\mathcal{C}} t_i(G)$$

In the following we study three optimization problems:

- MinPeriod: find a plan G that minimizes the period;
- MinLatency: find a plan G that minimizes the latency;
- 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 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-Hom is the problem of minimizing the period on homogeneous platforms while MinLatency-Het is the problem of minimizing the latency on heterogeneous platforms.

3 Related work

The main reference for this work is a recent paper by Srivastava, Munagala and Burge [14]. In fact, we utilize the very same application framework and execution model as those of [14]. Therefore, we refer the reader to [14], and to the many references therein, for the motivations of this study. In a word, applications include all domains where clients need to query multiple web services simultaneously, in a transparent and integrated fashion. The only difference with [14] is that we consider different-speed servers in addition to identical servers. Because web servers are very likely to be heterogeneous, this extension appears to be very natural and important. We aim at assessing the additional difficulty introduced by server heterogeneity, from both a theoretical perspective (deriving new complexity results) and a practical perspective (designing efficient heuristics).

Srivastava, Munagala and Burge [14] study the MinPeriod-Hom problem. For independent services, they characterize the structure of an optimal plan: a linear chain composed of the services whose selectivity does not exceed 1, arranged per non decreasing costs, followed by a fork including all services whose selectivity is larger than 1. Our first question was to assess the impact of introducing different-speed servers: what is the complexity of MinPeriod-Het? We show that this problem is indeed NP-hard.

The authors of [14] also consider services with dependencies. They propose an optimal polynomial algorithm for the problem, which is based on an integer linear formulation of the problem. The fact that this latter problem can be solved in polynomial time, due to the particular nature of the constraints, is shown in [12]. With dependence constraints, the problem MinLatency-Hom is shown NP-hard in [12]. They even prove that MinLatency-Hom with dependencies is as hard to approximate as the densest k-subgraph problem. We show in this paper that MinLatency-Hom has polynomial complexity when services are independent, and that MinLatency-Het with independent services is NP-hard.

---

so on.

Paper [13] studies the same model of filters in a different context: the authors consider a fixed chain of m servers of increasing speeds. Between two successive servers, there is a network link with a certain transmission cost. The model consists in partitioning the set of filters into m subsets, where each subset corresponds to the services that will be processed on one of the m servers. The authors aim at finding a mapping which optimizes the latency. They present a polynomial time algorithm for this problem. The problem is NP-hard when filters are correlated, which means that the selectivity of a filter depends of its ancestors. A 4-approximation is presented. The model is extended to a tree of servers: data is acquired at the leaves of the tree; for each internal node of the tree, a process makes the join with a certain cost and a certain selectivity depending upon its in-degree. A polynomial algorithm is presented in this case.

We point out that both [14] and [12] restrict to one-to-one mappings, but [13] uses other mapping rules. If we allow that several services can be mapped onto the same server, then it is easy to see that both MinPeriod-Hom and MinLatency-Hom become NP-hard. Indeed, in both cases, there is a direct reduction from the 2-Partition problem [9]: we create an instance of our problem in which the service costs are the numbers in the instance of 2-Partition and all selectivities are equal to 1, with two processors. We then ask whether there exists a plan whose period, or latency, is equal to half the sum of the numbers in the instance of 2-Partition.

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.

Several papers aim at mapping applications whose dependence graph is a linear pipeline: see [15, 16] for homogeneous platforms, and [3] for heterogeneous platforms. These papers use more general mapping rules than ours, but they do not deal with filters (in other words, all services have selectivity 1).

Many authors consider the problem of mapping communicating tasks onto heterogeneous platforms (which corresponds to a different type of scheduling problems, but with the same target platforms as in this paper). In [17], Taura and Chien consider applications composed of several copies of the same task graph, expressed as a DAG (directed acyclic graph). These copies are to be executed in pipeline fashion. Taura and Chien also restrict to mapping all instances of a given task type onto the same server. Their problem is shown NP-complete, and they provide an iterative heuristic to determine a good mapping. At each step, the heuristic refines the current clustering of the DAG. Beaumont et al [2] consider the same problem as Taura and Chien, i.e. with a general DAG, but they allow a given task type to be mapped onto several servers, each executing a fraction of the total number of tasks. The problem remains NP-complete, but becomes polynomial for special classes of DAGs, such as series-parallel graphs. For such graphs, it is possible to determine the optimal mapping owing to an approach based upon a linear programming formulation. The drawback with the approach of [2] is that the optimal throughput can only be achieved through very long periods, so that the simplicity and regularity of the schedule are lost, while the latency is severely increased.

Another important series of papers comes from the DataCutter project [7]. One goal of this project is to schedule multiple data analysis operations onto clusters and grids, decide where to place and/or replicate various components [4, 5, 11]. A typical application is a chain of consecutive filtering operations, to be executed on a very large data set. The task graphs targeted by DataCutter are more general than linear pipelines or forks, but still

more regular than arbitrary DAGs, which makes it possible to design efficient heuristics to solve the previous placement and replication optimization problems.

Finally, we point out that two recent papers [18, 19] target workflows structured as arbitrary DAGs and consider bi-criteria optimization problems on homogeneous platforms.

4 Period

In this section, we show that problem MinPeriod-Het is NP-complete. We provide a formulation in terms of an integer linear program, whose solution (on small problem instances) will be used to assess the absolute performance of the polynomial heuristics that we derive in Section 7.

4.1 General structure of optimal solutions

The following property was presented in [14] for homogeneous platforms. We extend it to different-speed servers:

Proposition 1. *Let $C_1, ..., C_n$, $S_1, ..., S_n$ be an instance of the problem* MinPeriod-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 3.*

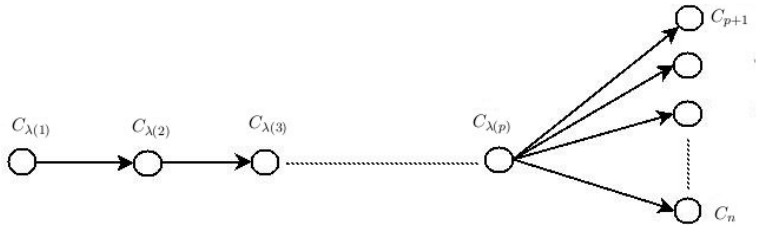

Figure 3: General structure

We point out that only the *structure* of the plan is specified by Proposition 1. 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 3), and to find the optimal assignment of services to servers.

Proof. In this proof, we denote the cost of a service C_i assigned to a server S_u in a plan G as:

$$cost_i(G) = \frac{c_i}{s_u} \prod_{(C_j \in \mathsf{Ancest}_i(G)} \sigma_j$$

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 a ancestor of C_j and that C_j is not a ancestor of C_i. Let $\mathsf{Ancest}_k(G)$ be the ancestors of C_k in G for all services k and $A'_k(G) = \mathsf{Ancest}_k(G) \cap \{C_1, ..., C_p\}$.

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 $C_j \in \mathsf{Ancest}_k(G)$, $\mathsf{Ancest}_k(G') = A'_k(G) \cup A'_i(G) \cup A'_j(G)$
- $\mathsf{Ancest}_i(G') = A'_i(G)$
- $\mathsf{Ancest}_j(G') = A'_j(G) \cup A'_i(G) \cup \{C_i\}$
- 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:

$$\begin{array}{rcl} cost_k(G') & = & \frac{c_k}{s_u} \times \prod_{C_l \in \mathsf{Ancest}_k(G')} \sigma_l \\ & = & \frac{c_k}{s_u} \times \prod_{C_l \in A'_k(G)} \sigma_l \\ & \leq & \frac{c_k}{s_u} \times \prod_{C_l \in \mathsf{Ancest}_k(G)} \sigma_l \\ & \leq & cost_k(G) \end{array}$$

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 3. □

4.2 NP-completeness of MinPeriod-Het

Theorem 1. MinPeriod-Het *is NP-complete.*

Proof. Consider the decision problem associated to MinPeriod-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 [20, 21]. 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 $\sum_{i=1}^{n} A[i] = n(n+1)$, otherwise we know that the instance has no solution. Then we point out that Equation 1 is equivalent to

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

We build the following instance $\mathcal{I}_2$ of MinPeriod-Het with n services and $p = n$ servers such that:

- $c_i = 2^{A[i]}$
- $\sigma_i = 1/2$

- $s_i = 2^i$
- $K = 2$

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 its plan is a linear chain (or we can transform it into such a chain, according to Proposition 1). 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. □

4.3 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 [14]. The structure of the solution is described in Section 4.1, 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).

4.4 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 [6], and the code is available in [8].

5 Latency

In this section, we study the problems MinLatency-Hom and MinLatency-Het. We reduce the structure of optimal plans for these problems to a certain class of graphs. We present a polynomial algorithm for the problem MinLatency-Hom and we show the NP-completeness of MinLatency-Het. For this latter problem we provide a formulation in terms of an integer linear program, and some complexity results for particular instances.

5.1 General structure of optimal solutions

Definition 1. *Given a plan G and a vertex $v = (C_i, S_u)$ of G,*

- *v is a leaf if it has no successor in G;*
- *$d_i(G)$ 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. *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 $d_{i_1}(G) = d_{i_2}(G)$, then 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 2.

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 $A_0 \cup ... \cup A_i$. Notice 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: $t_\ell(G') = t_\ell(G)$. And we have for each service C_k in A_0:

$$\begin{aligned} t_k(G') &= t_j(G') + \left(\prod_{C_\ell \in B_i} \sigma_\ell\right) \times c_k \\ &\leq t_j(G') + \left(\prod_{C_\ell \in B_i} \sigma_\ell\right) \times c_i \\ &\leq t_i(G') = t_i(G) \end{aligned}$$

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

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

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 $c_i/s_u > c_j/s_v$. 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 2 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. □

Corollary 1. *Let $C_1, ..., C_n, S_1, ..., S_n$ be an instance of* MinLatency-Het *and G an optimal plan for latency on this instance. We can compute in polynomial time a graph G' that respects properties of Proposition 2.*

Proof. In the proof of Proposition 2, we have seen a method to transform a graph in a graph that respects properties of Proposition 2. In this method, we execute many steps of transformation. One step can be computed in time $O(n^2)$ and there is at most $O(n^3)$ steps of tranformation. □

5.2 Polynomial algorithm on homogeneous platforms

In this section, we describe an optimal algorithm for MinLatency-Hom.

Data: n 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 t_j of C_i in G with predecessors C_1, ..., C_j;
    end
    Choose j such that t_j = min_k{t_k};
    Add the node C_i and the edges C_1 → C_i, ..., C_j → C_i to G;
end
```

Algorithm 1: Optimal algorithm for latency on homogeneous platforms.

Lemma 1. *Algorithm 1 verifies the following properties:*

- $t_1(G) \leq t_2(G) \leq \cdots \leq t_p(G)$
- $\forall i \leq n$, $t_i(G)$ *is optimal*

Proof. Let $C_1, ..., C_n$ be an instance of MinLatency-Hom with $c_1 \leq \cdots \leq c_p$, $\sigma_1, ..., \sigma_p \leq 1$ and $\sigma_{p+1}, ..., \sigma_n > 1$. Let G be the graph produced by Algorithm 1 on this instance. We prove by induction on i that every $t_i(G)$ is optimal and that $t_1(G) \leq t_2(G) \leq \cdots \leq t_p(G)$.

C_1 has no predecessors in G, so $t_1(G) = c_1$. Suppose that there exists G' such that $t_1(G') < t_1(G)$. If C_1 has no predecessor in G', then $t_1(G') = c_1 = t_1(G)$. Otherwise, let C_i be a predecessor of C_1 such that C_i has no predecessors. $t_1(G') > c_i \geq c_1$. In both cases, we obtain a contradiction with the hypothesis $t_1(G') < t_1(G)$. So $t_1(G)$ is optimal.

Suppose that for a fixed $i \leq p$, $t_1(G) \leq t_2(G) \leq \cdots \leq t_{i-1}(G)$ and $\forall j < i$, $t_j(G)$ is optimal. Suppose that there exists G' such that $t_i(G') < t_i(G)$ and $t_i(G')$ is optimal. Let C_k be the predecessor of C_i of greatest cost in G'. If $c_k < c_i$, then by hypothesis, $t_i(G') = t_i(G)$. Otherwise, if we choose for C_i the same predecessors than for C_k, we strictly reduce $t_i(G')$. Suppose that $c_k > c_i$. Let C_j be a predecessor of C_i in G' such that $c_j > c_i$ and for all $C_l \in \mathsf{Ancest}_j(G')$, $c_l < c_i$. Let C_m the predecessor of C_j of maximal cost. We have

$$\begin{array}{rcl} t_i(G') & \geq & t_j(G') \\ & \geq & \max_{C_\ell \in \mathsf{Ancest}_j(G')} t_\ell(G') + \left(\prod_{C_\ell \in \mathsf{Ancest}_j(G')} \sigma_l\right) c_j \\ & \geq & \max_{C_\ell \in \mathsf{Ancest}_j(G')} t_\ell(G) + \left(\prod_{C_\ell \in \mathsf{Ancest}_j(G')} \sigma_l\right) c_j \\ & \geq & \max_{\ell \leq m} t_\ell(G) + \left(\prod_{\ell \leq m} \sigma_\ell\right) c_j \\ & \geq & \max_{\ell \leq m} t_\ell(G) + \left(\prod_{\ell \leq m} \sigma_\ell\right) c_i \\ & \geq & t_i(G) \end{array}$$

However, $t_i(G')$ is optimal. So, we obtain a contradiction. $t_i(G)$ is optimal.

Suppose that for a fixed $i > p$, $\forall j < i$, $t_j(G)$ is optimal and $t_1(G) \leq t_2(G) \leq \cdots \leq t_p(G)$. For all $k > p$, we have

$$\begin{array}{rcl} \max_{j \leq k} t_j(G) + \prod_{j \leq k} \sigma_j * c_i & = & \max_{j=p}^{k} t_j(G) + \prod_{j=1}^{k} \sigma_j * c_i \\ & \geq & t_p(G) + \prod_{j \leq k} \sigma_j * c_i \\ & > & t_p(G) + \prod_{j \leq p} \sigma_j * c_i \end{array}$$

This relation proves that in G, the service i has no predecessor of selectivity strictly greater than 1. Suppose that there exists G' such that $t_i(G') < t_i(G)$ and $t_i(G')$ is optimal. Let C_k be the predecessor of C_i in G' of greatest cost. $\mathsf{Ancest}_i(G') \in \{1, k\}$. We obtain

$$\begin{array}{rcl} t_i(G') & = & t_k(G') + \left(\prod_{C_j \in \mathsf{Ancest}t_i(G')} \sigma_j\right) c_i \\ & \geq & t_k(G') + \left(\prod_{j \leq k} \sigma_j\right) c_i \\ & \geq & t_i(G) \end{array}$$

So, we obtain a contradiction. $t_i(G)$ is optimal.

For $i \leq p$, suppose that $t_i(G) < t_{i-1}(G)$. 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

$$\begin{array}{rcll} t_{i-1}(G'') & = & \max_{k \leq j} t_k(G) + \prod_{k \leq j} \sigma_k c_{i-1} & \text{by construction of node } C_{i-1} \\ & \leq & \max_{k \leq j} t_k(G) + \prod_{k \leq j} \sigma_k c_i = t_i(G) & \end{array}$$

However, $t_{i-1}(G)$ is optimal, and so $t_{i-1}(G) \leq t_{i-1}(G'') \leq t_i(G)$, which leads to a contradiction. Therefore we have $t_1(G) \leq t_2(G) \leq \cdots \leq t_p(G)$. □

Theorem 2. *Algorithm 1 computes the optimal plan for latency.*

Proof. The latency of the output plan G is the completion time of C_i such that $\forall C_j \quad t_i(G) \geq t_j(G)$. So, for all plan G', $\mathcal{L}(G') \geq t_i(G') \geq t_i(G) = \mathcal{L}(G)$. So, $\mathcal{L}(G)$ is the optimal latency. □

Algorithm 1 computes the optimal latency in $O(n^2)$.

5.3 NP-completeness of MinLatency-Het

Lemma 2. *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 2 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 2. Suppose that there exists a distance to leaves d such that the nodes at this distance do not respect Property 2 of Lemma 2. 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 all node $(C_i, S_u) \notin A$, $\mathsf{Ancest}_i(G') = \mathsf{Ancest}_i(G)$
- For all 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 $\sigma = \sum_{C_j \in \mathsf{Ancest}_i(G)} \sigma_j$.

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

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, 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 2 than in G. In addition, G' respect properties of Proposition 2. By induction, we can obtain a graph as described in Lemma 2. □

Theorem 3. MinLatency-Het *is NP-complete.*

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 $A = (A[1], \ldots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that

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

We can suppose that $\sum_{i=1}^{n} A[i] = n(n+1)$. We build the following instance $\mathcal{I}_2$ of Min-Latency-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 2. 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 2. 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 $(C_i, S_{\lambda_1}(i))$,

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

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. □

5.4 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 $A = (A[1], \ldots, A[n])$ of size n, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that

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

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

$$\frac{2^{u_1} + 2^{u_2} + \cdots + 2^{u_n}}{n} \geq 2^{\frac{u_1+u_2+\cdots+u_n}{n}} \tag{5}$$

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 (5). 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 4.

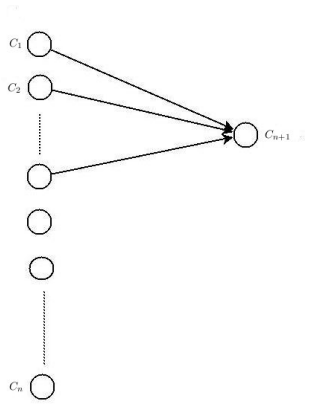

Figure 4: When the optimal plan is a star graph.

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
- $(C_i, S_j) \in V$ 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 5.

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

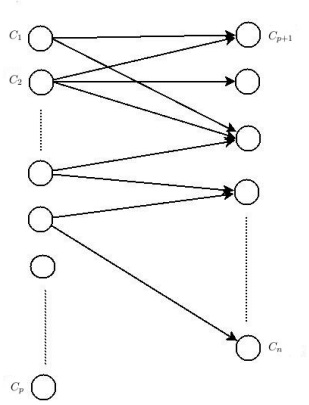

Figure 5: When the optimal plan is a bipartite graph.

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.

5.5 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 every server, there is exactly one service with exactly one set of predecessors:

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

- Every 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:

$$\forall i, i' \in \mathcal{C}, \forall u, u' \in \mathcal{S}, \forall e, e' \subset \mathcal{C}, e \nsubseteq e', i \in e', \quad z(i,u,e) + z(i',u',e') \leq 1$$

- 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 Bi-criteria problem

We only study the bi-criteria optimization problem in the homogeneous case, and we present a polynomial time algorithm for this problem. In the heterogeneous case, the bi-criteria problem is NP-complete since MinPeriod-Het and MinLatency-Het are NP-complete.

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 throughput K

Result: a plan G optimizing the latency with a throughput less than K

```
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 t_j of C_i in G with predecessors C_1, ..., C_j;
    end
    Let S = {k | c_i ∏_{0≤k<i} σ_k ≤ K};
    Choose j such that t_j = min_{k∈S}{t_k};
    Add the node c_i and the edges C_1 → C_i, ..., C_j → C_i to G;
end
```

Algorithm 2: Optimal algorithm for latency with a fixed throughput.

Proposition 3. *Algorithm 2 computes the optimal latency for a bounded period.*

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

7 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 8, the absolute performance of these heuristics will be assessed through a comparison with the (optimal) solution returned by the integer linear program of Section 4.4. We do not produce heuristics nor experiments for MinLatency-Het, because the integer linear

program of Section 5.5 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 [8].

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[2]. 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.

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.

[2]The interested reader will find a fully detailed proof for a very similar mapping problem in [10].

8 Experiments

Several experiments have been conducted for MinPeriod-Het in order to assess the performance of the heuristics described in Section 7.

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.

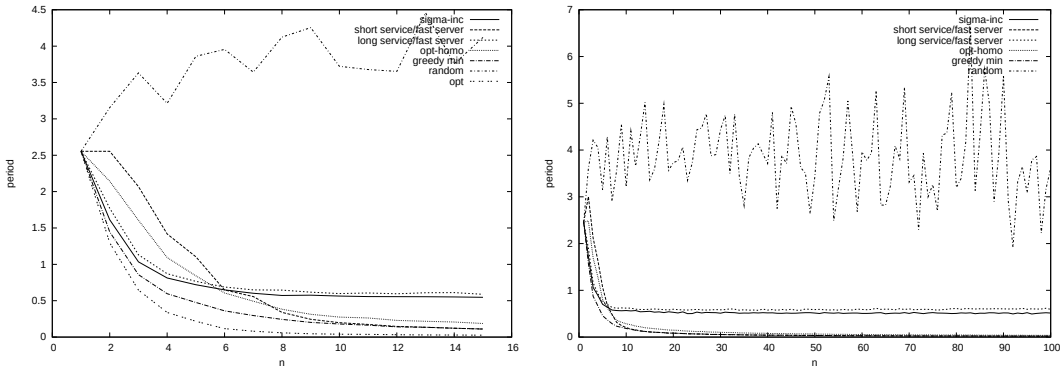

Figure 6: Experiment 1: general experiment.

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.

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

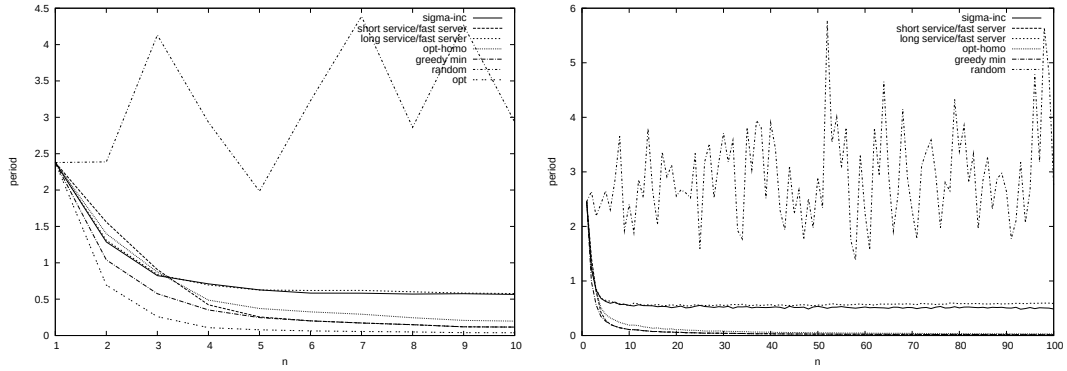

Figure 7: Experiment 2: with small selectivities.

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.

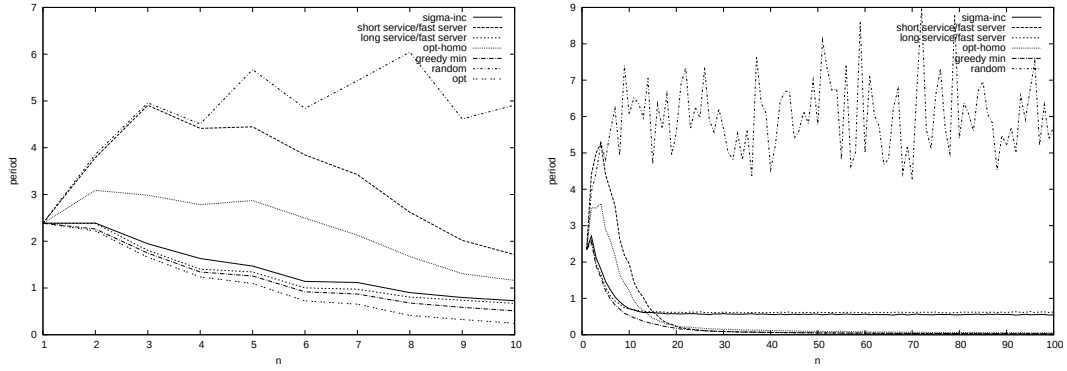

Figure 8: Experiment 3: with big selectivities.

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 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.

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

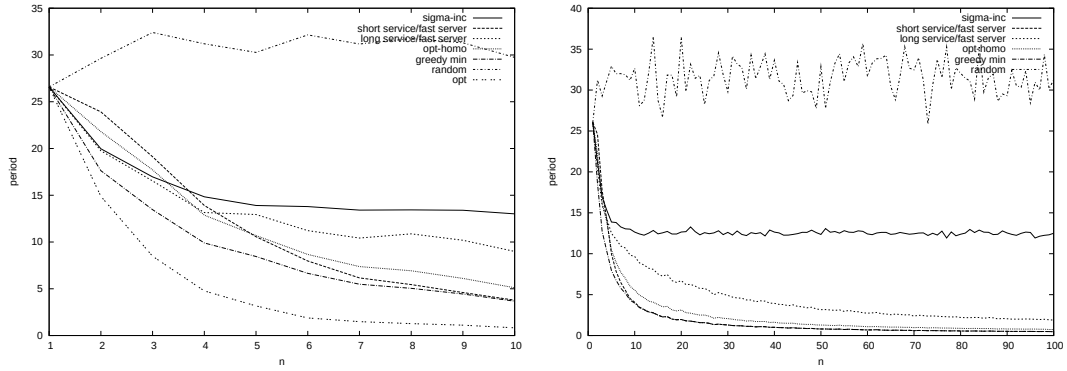

Figure 9: Experiment 4: with high heterogeneity.

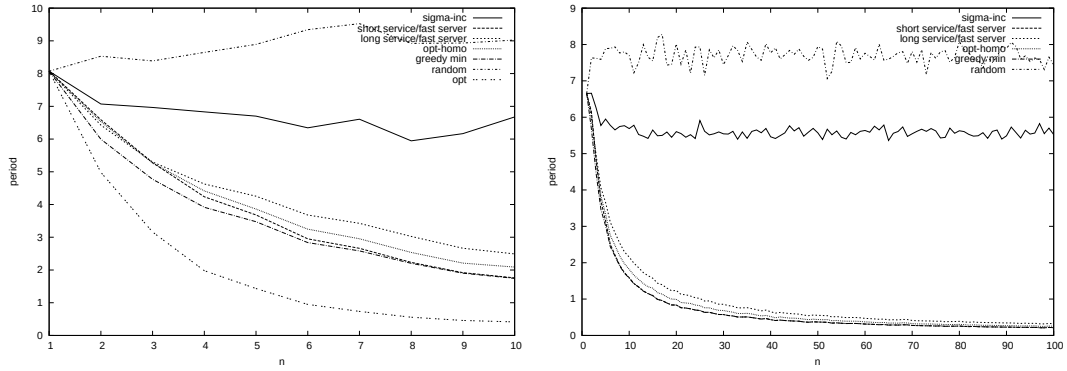

Figure 10: Experiment 5: with small heterogeneity.

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 11 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.

9 Conclusion

In this paper, we study the problem of one-to-one mappings of filters onto homogeneous and heterogeneous platforms. We study the complexity of the problem for the optimization of two different criteria, the latency and the period. We provide a polynomial time algorithm for MinLatency-Hom and we prove the NP-completeness of MinPeriod-Het and of MinLatency-Het. We also provide a polynomial time algorithm on homogeneous platforms for optimizing the latency given a threshold period. We present many heuristics and a linear program for the problem MinPeriod-Het.

As future work, we still need to design heuristics for MinLatency-Het and to find

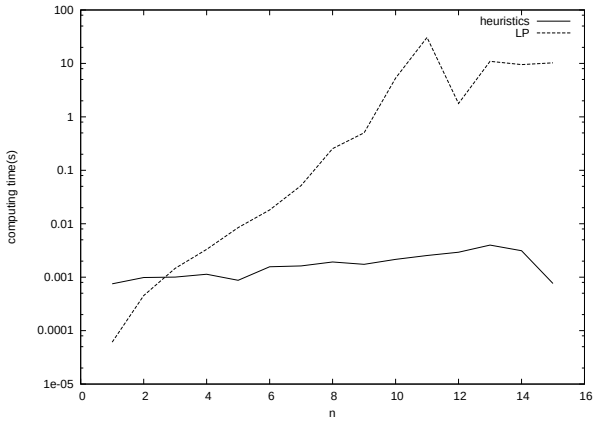

Figure 11: Computing times

a way to assess their performance (recall the integer linear program of Section 5.5 is untractable). Also, the model used in this work could be extended to more realistic settings by taking communication costs into account. A natural extension of this work would be to choose a model for the communication costs, and to study the complexity in the case of homogeneous platforms, since the problem without communications can be solved in polynomial time. It would also be challenging to design polynomial time heuristics which account for communications, and to perform some real experiments to assess the performance of these heuristics.

References

[1] M. P. Alessandro Agnetis, Paolo Detti and M. S. Sodhi. Sequencing unreliable jobs on parallel machines. *Journal on Scheduling*, May 2008.

[2] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Assessing the impact and limits of steady-state scheduling for mixed task and data parallelism on heterogeneous platforms. In *HeteroPar'2004: International Conference on Heterogeneous Computing, jointly published with ISPDC'2004: International Symposium on Parallel and Distributed Computing*, pages 296–302. IEEE Computer Society Press, 2004.

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

[4] M. Beynon, A. Sussman, U. Catalyurek, T. Kurc, and J. Saltz. Performance optimization for data intensive grid applications. In *PProceedings of the Third Annual International Workshop on Active Middleware Services (AMS'01)*. IEEE Computer Society Press, 2001.

[5] M. D. Beynon, T. Kurc, A. Sussman, and J. Saltz. Optimizing execution of component-based applications using group instances. *Future Generation Computer Systems*, 18(4):435–448, 2002.

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

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

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

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

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

[11] M. Spencer, R. Ferreira, M. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz. Executing multiple pipelined data analysis operations in the grid. In *2002 ACM/IEEE Supercomputing Conference*. ACM Press, 2002.

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

[13] U. Srivastava, K. Munagala, and J. 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.

[14] 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*, pages 355–366. VLDB Endowment, 2006.

[15] J. Subhlok and G. Vondran. Optimal mapping of sequences of data parallel tasks. In *Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'95*, pages 134–143. ACM Press, 1995.

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

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

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

[19] N. Vydyanathan, U. Catalyurek, T. Kurc, P. Saddayappan, and J. Saltz. Optimizing latency and throughput of application workflows on clusters. Research Report OSU-CISRC-4/08-TR17, Ohio State University, Columbus, OH, Apr. 2008. Available at ftp://ftp.cse.ohio-state.edu/pub/tech-report/2007.

[20] W. Yu. *The two-machine flow shop problem with delays and the one-machine total tardiness problem*. PhD thesis, Technishe Universiteit Eidhoven, June 1996.

[21] 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*, 7(5):333–348, 2004.