Mapping Filtering Streaming Applications With Communication Costs

Kunal Agrawal
CSAIL, Massachusetts Institute of Technology
USA
kunal_ag@mit.edu

Anne Benoit, Fanny Dufossé,
and Yves Robert
École Normale Supérieure de Lyon, France
Lyon, France
Anne.Benoit@ens-lyon.fr,
Fanny.Dufosse@ens-lyon.fr,
Yves.Robert@ens-lyon.fr

ABSTRACT

In this paper, we explore the problem of mapping *filtering streaming applications* on large-scale homogeneous platforms, with a particular emphasis on communication models and their impact. Filtering application are streaming applications where each node also has a *selectivity* which either increases or decreases the size of its input data set. This selectivity makes the problem of scheduling these applications more challenging than the more studied problem of scheduling "non-filtering" streaming workflows. We identify three significant realistic communication models. For each of them, we address the complexity of the following important problems:
- Given an execution graph, how can one compute the period and latency? A solution to this problem is an operation list which provides the time-steps at which each computation and each communication occurs in the system.
- Given a filtering workflow problem, how can one compute the schedule that minimizes the period or latency? A solution to this problem requires generating both the execution graph and the associated operation list.

Altogether, with three models, two problems and two objectives, we present 12 complexity results, thereby providing solid theoretical foundations for the study of filtering streaming applications.

Categories and Subject Descriptors

F.2.2 [**Analysis of algorithms and probem complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*; F.1.2 [**Computation by abstract devices**]: Modes of Computation—*Parallelism and concurrency*

General Terms

Algorithms, Theory

SPAA'09, August 11–13, 2009, Calgary, Alberta, Canada.

Keywords

query optimization, web service, streaming application, workflow, communication model, period, latency, complexity results

1. INTRODUCTION

This paper addresses the problem of mapping *filtering streaming applications*, or *filtering workflows*, on parallel platforms. This mapping problem was first studied in the context of query optimization over web services: in [17, 8], the authors consider the case where the web services were to be mapped one-to-one (one service on one processor) onto identical servers. A recent extension considers the same problem with different-speed servers [5]. However, this work does not model communication, and does not include communication cost in the analysis. In this paper, we introduce three different and realistic communication models, and we revisit the problem in this more challenging framework.

Filtering workflows resemble traditional pipelined workflows, a popular programming paradigm for streaming applications like video and audio encoding and decoding, DSP applications etc [10, 18, 22]. Filtering workflows consist of a collection of services that must be applied on a stream of consecutive data sets. These services are represented using a workflow graph, which contains several *nodes* (the services), and these nodes are connected to each other using first-in-first-out *channels* (or dependence constraints between services). Data is input into the graph using input channel(s) and the outputs are produced on the output channel(s). So far, the filtering workflows resemble regular workflows. However, in filtering workflows, the services *filter* incoming data by a certain amount. More precisely, each service C_i has a *selectivity* σ_i and an *elementary cost* c_i. If an input of size δ_i is provided to C_i, then the output is of size $\sigma_i\delta_i$ while the computation requirement of the service is $c_i\delta_i$. Therefore, when $\sigma_i < 1$, the service shrinks data (hence the name *filter*) and when $\sigma > 1$, the service expands the data. Just as in regular workflows, the goal is to map these computations on some parallel platform and then organize computations and communications so as to optimize some scheduling objective. Since data continually flows through these applications, typical objectives of the scheduler are to minimize the *period* (which is defined as the inverse of the throughput) or the *latency* (also called response time) [19, 20, 6, 21].

In this paper, we only consider mapping filtering work-

flows on *homogeneous machines*: each server has the same speed, and all servers are connected to each other by communication links of equal bandwidth. As in previous work [17, 8], we consider *one-to-one* mappings, where each server has at most one service mapped to it. Clearly, for one-to-one mappings, the number of servers must be equal to or more than the number of services. We show in this paper that most period or latency minimization are NP-complete even in this setting[1]. Note that we do not need to specify which service is mapped onto which server, since all servers are equivalent and we only map one service to a server.

Instead, given a collection of services, we specify a *plan*, which is the combination of an *execution graph* and an *operation list*:

- The execution graph is a directed acyclic graph (DAG) of services. If there is an edge from service C_i to C_j in the execution graph, then output of service C_i is sent to service C_j. The set of edges in the execution graph must obey the original precedence constraints but may also include additional edges to further filter the incoming data of some services, thereby reducing their actual execution time.
- The operation list describes the time-steps at which every computation and every communication takes place. We assume that the schedule is cyclic, so that the execution list can be specified concisely.

For one-to-one mappings on a homogenous platform, a plan fully describes a solution. Therefore, in order to minimize period or latency, we just have to find the corresponding execution graph and an operation list that minimizes our objective.

Let δ_0 be the size of the input data sets to the services which have no predecessors. The size of the input data set to a service C_j is δ_0 times the product of the selectivities of all of its ancestors. Therefore, in some cases, it may be advantageous to add an edge (in the execution graph) that did not exist in the original precedence constraints. For instance, if we add an edge from C_i to C_j, where C_i has a small selectivity, then C_j will require a smaller computation time and have a smaller output. However, we can not add edges arbitrarily since adding edges may increase communication costs; for our example, C_i now has an extra communication link to C_j due to the added edge.

All of the assumptions related to service costs and selectivities are those of Srivastava et al. [17, 8]. Although their papers mainly deal with query optimization over web services (already an increasingly important application with the advent of Web Service Management Systems [11, 15]), the approach applies to general data streams [3] and to database predicate processing [9, 13]. In addition, our framework is quite similar to the problem of scheduling unreliable jobs on parallel machines [1] where service selectivities correspond to job failure probabilities. Due to lack of space, please refer to [4] for more related work.

The emphasis of our work is on the impact of communication models. We consider two commonly used communication models. The *no overlap* communication model requires that at any point, a server can either compute, or receive an incoming communication, or send an outgoing communication. This models single threaded machines where every operation is serialized. We define two variants for this model, one where we enforce in-order execution, and another where we allow out-of-order execution (which means interleaving communications and computations of different data sets) so as to reduce the idle-time incurred by the serial ordering of the communications. The first variant is conservative (in the sense that it can always be implemented without additional resource requirement) while the second variant may require large buffering capacities. In contrast to these variants which sequentialize operations, the *overlap* communication model considers the situation where a server can compute and send/receive communications at the same time, and represents multi-threaded machines. In all models, communication is synchronous and we do not allow preemption (interruption) of either computation or communication.

Our main result is that computing the period or the latency in all these models turns out to be difficult. As already stated, the minimization problems (finding the optimal plan to minimize the period or the latency) are all NP-hard. This result is surprising, since polynomial algorithms exist for homogeneous machines when we do not model communication [17, 8]. Therefore, modeling communication costs explicitly has a huge impact on the difficulty of mapping filtering services. In addition, and quite unexpectedly, the "orchestration" problems (given an execution graph, find the optimal operation list) also are of combinatorial nature. Finally, the choice of the model has a tremendous impact on the values that can be achieved. Many of our results and counter-examples apply to regular workflows (without selectivities), and should be of great interest to the whole community interested in scheduling streaming applications.

This paper is organized as follows. Section 2 describes the framework of the problem in more details. Section 3 illustrates the difference between communication models with the help of several examples. The next two sections constitute the core of the paper. Section 4 is devoted the period minimization problem, while Section 5 is the counterpart for latency minimization. Finally we give some conclusions and perspectives in Section 6.

2. FRAMEWORK

This section is devoted to a precise statement of the different models and optimization problems. We then give a formal definition of the period and of the latency. Surprisingly, these formal definitions require quite a complicated formulation, so we work out an example in full details, in order to illustrate the differences between all the models.

2.1 Plans

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

The target machine is a homogeneous platform with p servers (or processors) of same speed s. All servers are connected to each other by communication links of equal bandwidth b. The cost for transmitting a data of size δ is $\frac{\delta}{b}$. Let δ_0 be the size of input data.

We have to build a *plan* $PL = (EG, OL)$, that is an ex-

[1]In general mappings, we can map several services onto the same server. Problems with general mappings are straightforwardly shown NP-hard by reduction from 2-Partition or bin packing [12].

ecution graph $EG = (\mathcal{C}, \mathcal{E})$ that summarizes all precedence relations in the mapping, and an operation list OL that captures the occurrence of each computation and each communication. We deal with the operation lists later, after having described the communication models. As for the execution graph $EG = (\mathcal{C}, \mathcal{E})$, the nodes in $\mathcal{C}$ are the services in $\mathcal{F}$ and input/output nodes. 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. Let $\mathsf{Ancest}_j(EG)$ denote the set of all ancestors[2] of C_j in the execution graph EG. 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 $C_i \in \mathsf{Ancest}_j(EG)$ [3].

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

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

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

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

Here, $C_{\text{in}}(k)$ is a lower bound of the time needed to receive input data from all the predecessors of C_k. The input data from each predecessor C_i is of size $\delta_0 \prod_{C_j \in \mathsf{Ancest}_i(EG)} \sigma_j$, hence it requires $\frac{\delta_0}{b} \prod_{C_j \in \mathsf{Ancest}_i(EG)} \sigma_j$ time units for communication from C_i. We add the communication from all the parents (immediate predecessors) to get the total incoming communication time $C_{\text{in}}(k)$. This lower bound may not be met because of idle times due to server synchronizations for the communications. However, we have not yet specified in which order the different communications take place. This specification requires discussion of communication models. We discuss variations of both one-port [7] and multi-port models [14] in Section 2.2.

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

- The selectivity of a service influences the execution time of *all* its successors (if any) in the mapping. In other words, a service is "filtered" or "expanded" by the combined selectivity of all its predecessors. This implies that selectivities are independent, and that the cost of join operations (in front of the service, after it receives all its communication) is negligible. All these hypotheses are enforced in the literature, but further work could be devoted to generalizations of this simple model.
- Since our platform is homogeneous, we can scale all service costs as $c_k \leftarrow \frac{b}{\delta_0} \cdot \frac{c_k}{s}$, allowing us to set $\delta_0 = b = s = 1$ without loss of generality. At the end of the computation, we can scale the computed period and latency by the factor $\frac{\delta_0}{b}$ to obtain the actual values.

[2]The ancestors of a service are the services preceding it, and the predecessors of their predecessors, and so on.

[3]Equivalently, $\mathcal{G}$ must be included, in the transitive closure of $\mathcal{E}$.

2.2 Communication models

This section presents the various communication models. We first present the general overview and then we formally state the constraints and the rules of the three models. We present only an informal description of the models. Detailed formulas are provided in [2].

With overlap.

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

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

The period $\mathcal{P}$ is defined as the interval between the completion of consecutive data sets. With this definition, the system can process data sets at a rate $1/\mathcal{P}$ (the throughput). In steady state, a new data set enters the system every $\mathcal{P}$ time-units, and several data sets are processed concurrently within the system. In the overlap model, the lower bound on the period is the maximum of the quantities $C_{\text{exec}}(k)$ over all services C_k:

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

Given an execution graph, it turns out that we can generate an order of communication/computations that achieves this lower bound in the multi-port model: see [2]. Note that determining the optimal execution graph is still NP-complete, and therefore, the period minimization problem is still difficult. See [2] for a complete list of the resource constraints that need to be satisfied for the OVERLAP model.

Without overlap.

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

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

From an architectural point of view, we emphasize that the InOrder and OutOrder variants may be overly pessimistic, as modern processors are capable of some internal parallelism. However, both operation modes correspond to blocking send/receive MPI primitives [16], and servers may encounter idle time due to the synchronizations in both models. Nevertheless, we expect less idle time for the OutOrder model than for the InOrder model, due to the additional schedule flexibility of the former model. Both variants lead to a computation cost for server/service C_k is bounded below by

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

As before, a lower bound on the period is the maximum of the execution times. But unlike the Overlap model with multi-port communications, this lower bound cannot always be reached: see the example in Section 2.3. Note that the multi-port model is more flexible: since it permits sending data to many other servers simultaneously, orchestrating the communications in the multi-port model is an easier task than for the one-port model. We refer to [2] for a list of resource constraints to be enforced for each model. We emphasize that there is no closed-form formula for the period with the InOrder and OutOrder models, which we believe is a new and surprising observation.

Latency.

We have just seen that models have a strong impact on the computation of the period. This is also true for the latency (or response time), but to a lesser extent. The latency (or response time) is the time needed to execute a single data set entirely. The overlap/no-overlap distinction is no longer meaningful for optimizing this criterion. Indeed, we can always fully serialize the processing of each data set and minimize the execution time, or makespan, when processing a unique data set. In other words, we delay the processing of the next data set until the current one is completely executed, this suppresses all resource conflicts. With such a strategy, the period is equal to the latency, which in turn is equal to longest path from an input node to an output node in the plan. However, the choice between one-port or multi-port communications does have an impact on the latency. This is illustrated by the example presented in Section 3.2.

Other variants.

Altogether, we have three models, one multi-port model with overlap and two one-port variants without overlap; the precise constraints that need be enforced are detailed in [2]. We note that other models can be introduced, for instance one-port communications with computation/communication overlap. However, we believe that we address the most realistic combinations: on single-threaded machines it is hard to avoid doing everything sequentially, and on multi-threaded machines, we can execute computations and (several) communications concurrently. Another possibility is to consider preemptive models where communication and/or computation can be interrupted, and the bandwidth of communication can vary during the communication. Such preemptive models are beyond the scope of this paper.

We point out that the effective difference between one-port and multi-port communications is not obvious: in most cases, the optimal solution for the multi-port model obeys the one-port constraints. In Section 3, we present examples of execution graphs were the optimal latency and period for the multi-port model are strictly smaller than the optimal ones for the one-port model.

Characterizing solutions.

In this paper, we study two optimization problems: (i) MinPeriod: find a plan $PL = (EG, OL)$ that minimizes the period; and (ii) MinLatency: find a plan $PL = (EG, OL)$ that minimizes the latency. For each problem instance, independently of the model and of the objective function, the solution includes the execution graph EG that describes the set Ancest_i for each service C_i. But this graph alone does not give enough information to compute the schedule, i.e., the moment at which each operation takes place. We also need the complete list of the time-steps at which every communication or computation begins and ends. In this paper, we only consider cyclic schedules, that is, schedules that repeat for each data set. Therefore, the description of the operation list is polynomial (actually, quadratic) in the number of services.

Formally, we define the operation list OL as follows:

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

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

For every model, we have rules that must be satisfied by the operation list in order to have a valid schedule. These rules ensure that no resource constraint or model assumption is violated. For instance, in the InOrder model, $\mathsf{EndComm}(j,k)^n < \mathsf{BeginComm}(i,j)^{n+1}$ for all services i, j, k and all data sets, since all work for one data set must be done before starting work on another data set. See [2] for the complete list of rules.

Note that all models are non-preemptive: once initiated, a

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

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

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

2.3 Illustrative Example

In this section, we work out a simple example of filtering workflow in order to better understand the three models. Consider an instance with 5 services, all of which have cost 4 and selectivity 1, without dependence constraints. Let the execution graph EG be the graph presented in Figure 1.

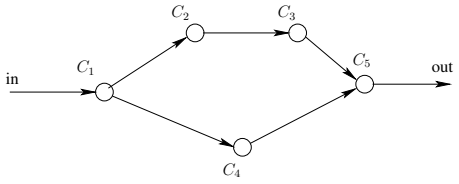

Figure 1: Example.

Latency.

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

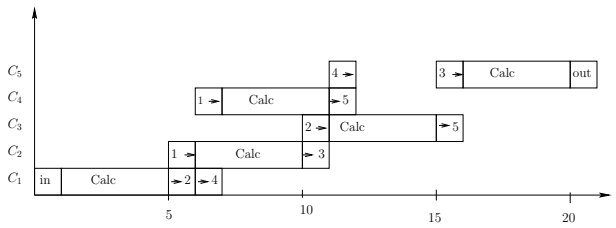

Figure 2: Execution scheme.

This execution scheme is presented in figure 2 With multi-port communications we cannot achieve a better latency for this example, so we derive the same solution. See Section 3.2 for an example where the multi-port latency is smaller than the one-port latency.

Period.

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

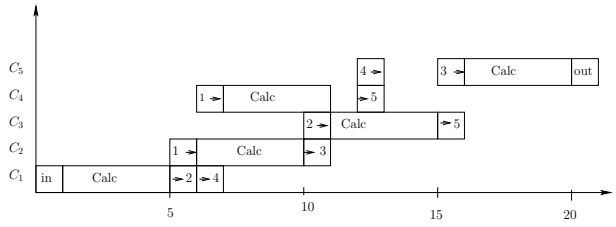

Figure 3: Optimal execution scheme for Overlap.

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

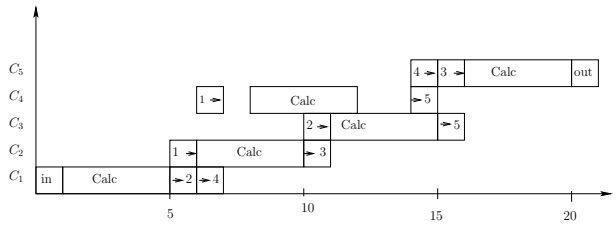

Figure 4: Optimal execution scheme for OutOrder.

For the model InOrder, we have the same lower bound for the period as for the model OutOrder, namely 7. With the previous operation list, we obtain a period 10 because of the cost of C_5: the beginning of the receive for data set 1 has

to wait for the end of the send of data set 0. This difference of 3 between 7 and 10 corresponds to the idle time between the end of the receive from C_4 and the beginning of the receive from C_3, which is the difference of the lengths of the path $C_1 \to C_4 \to C_5$ and of the path $C_1 \to C_2 \to C_3 \to C_5$. This idle time can be reduced by shareing it between C_1, C_4 and C_5 as follows. The time spent in computations and communications is 7 for C_1, 6 for C_4 and 7 for C_5 respectively. The optimal solution is to give an idle time $\frac{2}{3}$ for C_1, $1 + \frac{2}{3}$ for C_4 and $\frac{2}{3}$ for C_5. We obtain the following values: $\mathsf{BeginComm}^0_{(1,4)} = 6 + \frac{2}{3}$,$\mathsf{EndComm}^0_{(1,4)} = 7 + \frac{2}{3}$, $\mathsf{BeginCalc}^0_{(4)} = 7+\frac{2}{3}$, $\mathsf{EndCalc}^0_{(4)} = 11+\frac{2}{3}$, $\mathsf{BeginComm}^0_{(4,5)} = 13 + \frac{1}{3}$, and $\mathsf{EndComm}^0_{(4,5)} = 14 + \frac{1}{3}$. The other values do not change. We obtain a period $\frac{23}{3}$, which the reader may find surprising! The resulting execution scheme is presented in Figure 5.

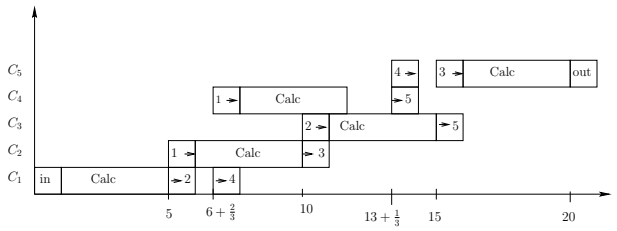

Figure 5: Optimal execution scheme for InOrder.

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

3. COUNTER-EXAMPLES

In this section we give three examples to show the difficulty introduced by communication costs with the different models.

3.1 With and without communication cost

Our first example shows the impact of communication costs on the optimal solution for the period: without communications, the optimal plan always is a linear chain for the services whose selectivities do not exceed 1 [17]. Here we will prove that this property is no longer true in the OVERLAP model.

Consider the following instance with 202 services: services C_1 and C_2 have selectivities $\sigma_i = 0.9999$ and costs $c_i = 100$, while services C_i for $3 \leq i \leq 202$ have selectivities $\sigma_i = 100$ and costs $c_i = \frac{100}{0.9999}$. For the model without communication cost, we obtain the optimal period of 100 by chaining the services with selectivity less than 1 (C_1 and C_2), and making C_2 the immediate predecessor of all other services. But because of the outgoing communications of C_2, this solution gives us a period 200 with the model OVERLAP. We claim that the only optimal solution with the model OVERLAP is the plan presented in Figure 6, which does not have the property that it chains the services with selectivity at most 1.

We prove by contradiction that this is only solution with period at most 100. Consider another plan with period less than 100. Let i be an integer with $3 \leq i \leq 202$. The cost

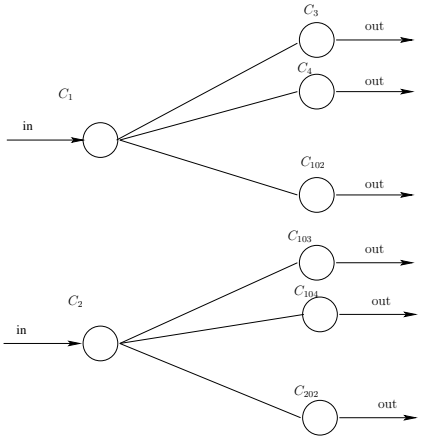

Figure 6: Optimal solution with communication costs.

of C_i is strictly greatest than 100, hence it must have C_1 or C_2 or both as a predecessor. But if C_1 precedes C_2, C_1 can only have 99 other successors, and C_2 can only have 100 successors (due to communication costs). There is a slot missing, unless some C_i has a successor C_j where $3 \leq i, j \leq 202$ and $i \neq j$. But then the computation time of C_j in G would be

$$\prod_{C_k \in \mathsf{Ancest}_j(G)} \sigma_k \times \sigma_i \times c_j \geq 0.9999^2 * 100 * 100 > 100$$

Therefore, the property of chaining all filters with "small" selectivities no longer works for models which have communication costs, making it more difficult to compute the plan with optimal period.

3.2 One-port/multi-port for latency

In this example, we study the difference between one-port and multi-port communications when computing the latency when the execution graph is given. Consider a problem instance with 12 services C_1 to C_{12}, all with unit cost. We assume that $\sigma_2 = \sigma_3 = 2$, $\sigma_4 = \sigma_5 = \sigma_6 = 3$, and the other selectivities are equal to 1. The execution graph EG is represented in Figure 7.

The computations of services C_1,...,C_6 can be completed at time 2. With multi-port communications, the communications between these services and services C_7,...,C_{12} can be executed within 6 time-steps; they all complete at time 8. The computations of services C_7,...,C_{12} complete at time 14 (the size of each input is 6), and the output communications to the outside world all complete at time 20, which leads to a latency $\mathcal{L} = 20$.

This latency cannot be achieved with one-port communications. To see this, first note that the computation of any service C_i with $1 \leq i \leq 6$ cannot be completed before time 2. Then the difference between the beginning of the computation of any service C_j with $7 \leq j \leq 12$ and the latency is at least 12, because of the cost of their computation and of their outgoing communication. Hence, to obtain a latency of 20, all communications from services C_1,...,C_6 to services C_7,...,C_{12} must be completed within 6 time-steps. But this value is equal to $C_{\text{out}}(i)$ for $1 \leq i \leq 6$ and to $C_{\text{in}}(j)$ for $7 \leq j \leq 12$: there cannot be any idle time on any ser-

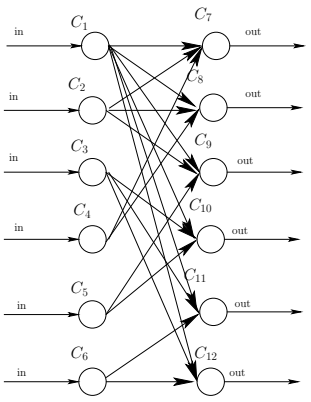

Figure 7: Execution graph for the example with the latency.

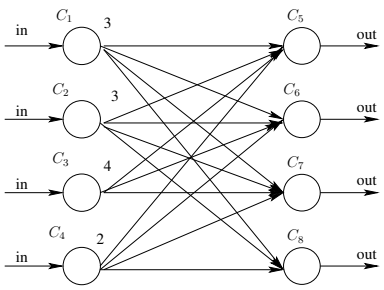

Figure 8: Execution graph for the example with the period.

vice between the first incoming data and the last outgoing data. Suppose that there exists a valid operation list for one-port communications without idle-time and capable of executing all the communications within 6 time steps. Let C_j be the service such that $\mathsf{BeginComm}^0_{(1,j)} = 2$ and let C_k be the service such that $\mathsf{BeginComm}^0_{(1,k)} = 3$. We know that these two services exist because there cannot be any idle time on C_1. Then C_k is necessarily idle between time 2 and 3 because the only incoming communication on C_k of cost 1 is $C_1 \rightarrow C_k$. This contradiction proves that the optimal latency with one-port communications on this instance is strictly greater than 20.

Note that this result also holds for traditional workflows (without selectivities), where the execution graph of the example can be viewed as the original DAG of a workflow where the weight of a node C_k is $C_{\text{comp}}(k)$ and where the volume of a communication from C_i to C_k is $\prod_{C_j \in \mathsf{Ancest}_i(EG)} \sigma_j$. To the best of our knowledge, this is a new and important observation for scheduling classical streaming applications.

3.3 One-port/multi-port for period

In this example, we study the difference between one-port and multi-port communications when computing the period, given the execution graph, in the Overlap model. The example with the period is more complicated than the one with the latency because different data sets can be processed concurrently. Consider the following problem instance with 8 services of small cost, so that communications are the bottleneck: $\forall i, c_i = 1/100$. We let $\sigma_1 = \sigma_2 = 3$, $\sigma_3 = 4$, $\sigma_4 = 2$, and $\sigma_i = 1/100$ for $5 \leq i \leq 8$ (output communications are negligible). The execution graph EG is represented in Figure 8.

With the multi-port model, the optimal value of the period is given by the maximum time needed for communications, i.e. $\mathcal{P} = 12$. Can we obtain this value with one-port communications? Notice that $C_{\text{out}}(1) = C_{\text{out}}(2) = C_{\text{out}}(3) = 12$ and $C_{\text{in}}(5) = C_{\text{in}}(6) = C_{\text{in}}(7) = 12$. That means that if there exists a solution, then there must be no idle time on these servers.

Suppose that there exists a valid operation list of period $\lambda = 12$. Consider the steady-state operation, and let t be a time-step at which a communication from C_3 to C_5 begins. Then, as there is no idle time on C_3 for outgoing communications, there is a communication from C_3 at time $t + 4$ (suppose it goes to C_6) and at $t + 8$ (suppose it goes to C_7).

There is no idle time on C_5 for incoming communications either, hence there is an incoming communication on C_5 that begins at time $t + 4$. This communication is of size 2 or 3. Suppose first that this communication is of size 3. This case is represented in Figure 9. We can suppose that it comes from C_1. We study the beginning time of the communication from C_1 to C_7. There is no idle on C_1, and this server sends data to C_5 between $t + 4$ and $t + 7$: hence this communication begins at time $t + 7$, $t + 10$ or $t + 13$ (or $t + 1$ for the previous data set). Server C_7 receive data from C_3 between $t + 8$ and $t + 12$. That means that the communication from C_1 cannot begin at $t + 7$, and nor at $t + 10$. There only remains time $t + 13$. There remains an idle slot between $t + 12$ and $t + 13$ for a communication, but there is no communication of size 1. As there should be no idle time on C_7, we obtain a contradiction.

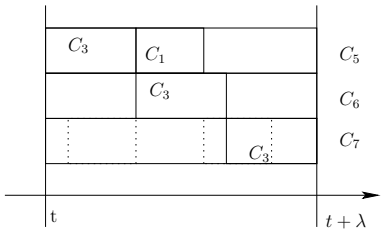

Figure 9: Case 1.

Suppose now that there is an incoming communication to C_5 of size 2 that begins at time $t + 4$, followed by a communication of size 3 (suppose it comes from C_1). We study the beginning time of the communication from C_1 to C_6. Server C_1 sends data to C_5 from time $t + 6$ to $t + 9$ and it has no idle time. Then the communication from C_1 to C_6 can begin at time $t + 9$ or $t + 12$ (or t for the previous data set) or $t + 15$ (or again $t + 3$ for the previous data set). This communication cannot begin at time $t + 3$ because it is of size 3 and the communication from C_3 to C_6 begins at time $t + 4$. If it begins at time $t + 9$, we obtain an idle time of size 1 between $t + 8$ and $t + 9$, and if it begins at time t, we obtain an idle time of size 1 between $t + 3$ and $t + 4$. However, we have seen in the previous case that this is not possible. We obtain a contradiction.

For minimizing the period in the Overlap model, we have

proven that multi-port communications are strictly "stronger" than one-port communications. Just as in Section 3.2 for the latency, we point out that this result still holds for traditional workflows (without selectivities).

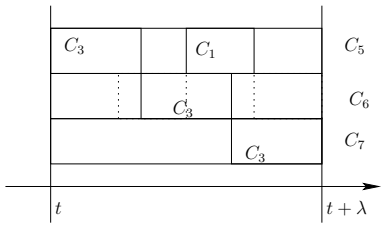

Figure 10: Case 2.

4. PERIOD MINIMIZATION

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

4.1 Optimal period for a given execution graph

THEOREM 1. *Given an execution graph, the problem of computing the operation list that leads to the optimal period has polynomial complexity with the* OVERLAP *model but is NP-hard with the* OUTORDER *and* INORDER *models.*

PROOF. Dealing with the OVERLAP model is not too difficult. In this model, all the communications can be executed in time

$$T = \max_{1 \leq k \leq n} \{C_{\text{in}}(k), C_{\text{out}}(k)\}$$

We just have to assign to any communication of size t a fraction t/T of the available bandwidth. Remember that we have normalized the bandwidth to $b = 1$. By doing so, the communications will be executed in time T, and the sum of incoming or outgoing communications on any server is less than or equal to $b = 1$. We have not yet specified which data sets are operated upon by the different servers. But the previous discussion shows that every server can repeat its operations every T time-units without conflict. If suffices to let the first data set traverse the execution graph greedily: each communication is performed as soon as possible, and each computation is performed as soon as all the necessary data (all incoming communication) is available. We then repeat this scheme for every data set every T time units, and we obtain an operation list of period T.

The NP-completeness reduction for non-overlapping models is rather involved. We present here a proof for the model INORDER. We consider the associated decision problem and show that is NP-complete: given an application $\mathcal{A} = (\mathcal{F}, \mathcal{G})$, an execution graph EG for this application, and a bound K, does there exist an operation list for EG such that the period does not exceed K? This problem is obviously in NP: given $\mathcal{A}$, EG and an operation list, we have the period λ and check whether it does not exceed K. To establish completeness, we use a reduction from RN3DM [23]. We consider an instance $\mathcal{I}_1$ of this problem: given an integer vector $A = (A[1], \ldots, A[n])$ of size $n \geq 2$, does there exist two permutations λ_1 and λ_2 of $\{1, 2, \ldots, n\}$ such that:

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

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 $\mathcal{I}_1$ has no solution. We associate to $\mathcal{I}_1$ an instance $\mathcal{I}_2$ with $2n+5$ services without dependence constraints, all of selectivity 1, and whose costs are as follows:

- $c_1 = c_{2n+5} = n$ and $c_{2n+3} = c_{2n+4} = 2n+1$
- $c_{2i} = 2n+1$ for $1 \leq i \leq n+1$ and $c_{2i+1} = 2n+1-A[i]$ for $1 \leq i \leq n$
- $\sigma_i = 1$ for $1 \leq i \leq 2n+5$

The execution graph is represented in Figure 11. Finally, we let $K = 2n+3$. The size of $\mathcal{I}_2$ is obviously linear in the size of $\mathcal{I}_1$. Intuitively, we note that service C_1 has many successors and C_{2n+5} many predecessors. We need the ordering of the associated communications to compute the optimal period for this execution graph. We now show that $\mathcal{I}_1$ has a solution if and only if $\mathcal{I}_2$ has a solution.

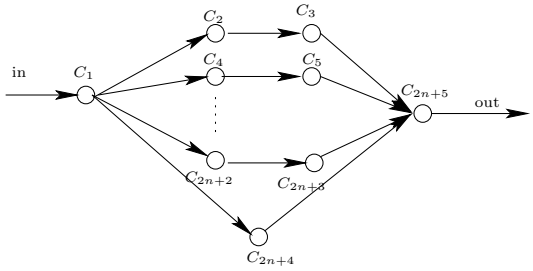

Figure 11: graph G.

Suppose first that $\mathcal{I}_1$ has a solution λ_1, λ_2. We compute the following operation list for $\mathcal{I}_2$: C_1 first communicates with C_{2n+4}. Then services C_2, C_4, ..., C_{2n} are fed in the ordering given by λ_1. Finally C_{2n+4} is the last service to receive data from C_1. Receptions by C_{2n+5} are done in the order C_1, $C_{2(n-\lambda_2(1))+3}$, ..., $C_{2(n-\lambda_2(n))+3}$, C_{2n+3}. With this orchestration, owing to Equation 1, the period is $2n+3$.

Suppose now that $\mathcal{I}_2$ has a solution. For a data set k, suppose that the computation of C_{2n+2} begins at time i and that the computation of C_{2n+4} begins at time j. For services C_1, C_{2n+2} and C_{2n+4}, the sum of the costs of communications and of computations is equal to $2n+3$. That means that there is no idle time for the associated servers. Hence at time $i-1$ (resp. $j-1$), there is a communication between servers C_1 and C_{2n+2} (resp. C_{2n+4}) for data set k. Hence service C_1 sends the result of its computation for data set k between time-steps $i-1$ and j or between time-steps $j-1$ and i. Therefore, $|j-i|+1 \leq n+2$. For services C_{2n+3}, C_{2n+4} and C_{2n+5}, the sum of the costs of communications and of computations is equal to $2n+3$. That

means that there is no idle time for the associated servers. Hence the computation of data set k on C_{2n+3} and C_{2n+4} are completed at time $i + 4n + 3$ and $j + 2n + 1$ respectively. C_{2n+5} receives the corresponding data from C_{2n+3} and C_{2n+4} at time $i + 4n + 3$ and $j + 2n + 1$ respectively. Then service C_{2n+5} receives the data for the computation of data set k between time $i + 4n + 1$ and $j + 2n$. Hence $|(i+4n+3)-(j+2n+1)|+1 = |(i-j)+2n+2|+1 \leq n+2$. We obtain $j - i = n + 1$. As a consequence, for $1 \leq i \leq n$, the communication from C_1 to C_{2i} is done between time j and $j+n$ and the communication between C_{2i+1} and C_{2n+5} is done between time $j+2n+2$ and $j+3n+2$. Let λ_1 be the ordering of communications from C_1 to services $C_2, ..., C_{2n}$ and λ_2 be the permutation such that $n + 1 - \lambda_2$ is the ordering of communication from $C_3, ..., C_{2n+1}$ to C_{2n+5}. We obtain

$\forall i, \lambda_1(i) + (2n + 1) + 1 + (2n + 1 - A[i]) + \lambda_2(i) = 4n + 3$
$\forall i, \lambda_1(i) + \lambda_2(i) = A[i]$.

This completes the proof for the InOrder model. The proof for the model OutOrder is provided in [2]. □

We should point out that Theorem 1 holds for regular streaming applications (without selectivities). This is an important and new result in that context.

4.2 Computing the optimal period

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

On a positive note, we derived the following result on the structure of the optimal execution graph: for any instance of MinPeriod without dependence constraints, and using any of the three models, there exists an optimal plan whose execution graph is a forest (see [2] for the proof). This "structural" result reduces the search of optimal execution graphs. Still, all minimization problems are NP-hard.

Theorem 2. *Problems* MinPeriod-Overlap, MinPeriod-OutOrder *and* MinPeriod-InOrder *without dependence constraints are all NP-hard.*

The proof of Theorem 2 is provided in [2]. Again, the NP-completeness reductions are quite involved.

We conclude this section by providing a particular polynomial instance of the problem: If we restrict the search for execution graphs and impose the restriction that the execution graph must be a linear chain, then the execution graph can be found in polynomial time using a greedy algorithm (see [2] for the proof). In addition, for this case, the operation list can be found quickly as well. Therefore, the problem of finding a plan which minimizes the period can be solved in polynomial time for all three communication models in this instance.

5. LATENCY MINIMIZATION

This section is the counterpart of Section 4 for the latency. First we address the following problem: given an execution graph, what is the complexity of determining the operation list that leads to the best latency? This problem turns out to be NP-hard for all models (while determining the best period was polynomial for the Overlap model). The general optimization problems MinLatency-Overlap, MinLatency-InOrder and MinLatency-OutOrder are all NP-hard. All these results imply technically involved reduction proofs.

5.1 Optimal latency for a given execution graph

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

Theorem 3. *Given an execution graph, the problem of computing the optimal operation list that leads to the optimal latency is NP-hard for the three models.*

The proof of Theorem 3 is provided in [2]. Also, we derive a polynomial case, namely computing the latency for a tree-shaped execution graph (see [2]). As for the period (Theorem 1), we point out that Theorem 3 holds for regular streaming applications (without selectivities). Again, this is an important and new result in that context.

5.2 Computing the optimal latency

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

Theorem 4. *Problems* MinLatency-Overlap, MinLatency-OutOrder *and* MinLatency-InOrder *without dependence constraints are all NP-hard.*

The proof of Theorem 4 is provided in [2]. We conclude this section by providing the complexity of the problems where we impose the restriction that the execution graph must be a chain or a forest:

- The problem MinLatency when restricting to plans whose execution graphs are linear chains is polynomial for all models: see [2] for a proof.
- The problem MinLatency when restricting to plans whose execution graphs are forests is NP-hard for all models: see [2] for a proof.

6. CONCLUSION

In this paper, we have explored the problem of mapping filtering streaming applications on large-scale homogeneous platforms, with a particular emphasis on communication models and their impact. We have identified three natural and realistic communication models, with and without

communication/computation overlap, and with one-port or bounded multi-port communications. We have addressed the following important problems:

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

We have been able to provide the complexity of all the 12 optimization problems, thereby providing solid theoretical foundations for the study of filtering streaming applications. Several of our results apply to regular workflow applications, which broadens the scope and significance of our results to quite a large applicative framework.

In the future, we plan to explore models that allow preemption. This would require to carefully assess the cost of interruptions. Another important extension of this work would be to tackle bi-criteria problems: given a threshold period, what is the optimal latency? and conversely, given a threshold latency, what is the optimal period? All bi-criteria problems are trivially NP-hard (since mono-criterion problems already are) but we can search for approximation algorithms, or at least efficient heuristics.

Acknowledgment.

We thank the reviewers for their comments and suggestions, which greatly improved the final version of the paper. The work of Anne Benoit and Yves Robert is supported by the ANR *Stochagrid* project.

7. REFERENCES

[1] A. Agnetis, P. Detti, M. Pranzo, and M. S. Sodhi. Sequencing unreliable jobs on parallel machines. *Journal on Scheduling*, 12(1):45–54, 2009.

[2] K. Agrawal, A. Benoit, F. Dufossé, and Y. Robert. Mapping filtering streaming applications with communication costs. Research Report 2009-06, LIP, ENS Lyon, France, Feb. 2009.

[3] 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*, pages 407–418. ACM Press, 2004.

[4] A. Benoit, F. Dufossé, and Y. Robert. Mapping filter services on heterogeneous platforms. Research Report 2008-19, LIP, ENS Lyon, France, June 2008. Available at `graal.ens-lyon.fr/~abenoit/`.

[5] A. Benoit, F. Dufossé, and Y. Robert. On the complexity of mapping filtering services on heterogeneous platforms. Research Report 2008-30, LIP, ENS Lyon, France, Oct. 2008. Short version to appear in IPDPS'09.

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

[7] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63:251–263, 2003.

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

[9] S. Chaudhuri and K. Shim. Optimization of queries with user-defined predicates. *ACM Trans. Database Systems*, 24(2):177–228, 1999.

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

[11] 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`.

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

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

[14] B. Hong and V. Prasanna. Bandwidth-aware resource allocation for heterogeneous computing systems to maximize throughput. In *Proceedings of the 32th International Conference on Parallel Processing (ICPP'2003)*. IEEE Computer Society Press, 2003.

[15] M. Ouzzani and A. Bouguettaya. Query processing and optimization on the web. *Distributed and Parallel Databases*, 15(3):187–218, 2004.

[16] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI the complete reference*. The MIT Press, 1996.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Q. Wu, J. Gao, M. Zhu, N. Rao, J. Huang, and S. Iyengar. On optimal resource utilization for distributed remote visualization. *IEEE Trans. Computers*, 57(1):55–68, 2008.

[22] 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.

[23] 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.