Offline and Online Scheduling of Concurrent Bags-of-Tasks on Heterogeneous Platforms

Anne Benoit, Loris Marchal, Jean-François Pineau, Yves Robert, Frédéric Vivien

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Offline and Online Scheduling of Concurrent Bags-of-Tasks on Heterogeneous Platforms

Anne Benoit — Loris Marchal — Jean-François Pineau — Yves Robert — Frédéric Vivien

N° ????

December 2007

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

Offline and Online Scheduling of Concurrent Bags-of-Tasks on Heterogeneous Platforms

Anne Benoit, Loris Marchal, Jean-François Pineau, Yves Robert, Frédéric Vivien

Thème NUM — Systèmes numériques
Projet GRAAL

Rapport de recherche n° ???? — December 2007 — 49 pages

Abstract: Scheduling problems are already difficult on traditional parallel machines. They become extremely challenging on heterogeneous clusters, even when embarrassingly parallel applications are considered. In this paper we deal with the problem of scheduling multiple applications, made of collections of independent and identical tasks, on a heterogeneous master-worker platform. The applications are submitted online, which means that there is no a priori (static) knowledge of the workload distribution at the beginning of the execution. The objective is to minimize the maximum stretch, i.e. the maximum ratio between the actual time an application has spent in the system and the time this application would have spent if executed alone.

On the theoretical side, we design an optimal algorithm for the offline version of the problem (when all release dates and application characteristics are known beforehand). We also introduce several heuristics for the general case of online applications.

On the practical side, we have conducted extensive simulations and MPI experiments, showing that we are able to deal with very large problem instances in a few seconds. Also, the solution that we compute totally outperforms classical heuristics from the literature, thereby fully assessing the usefulness of our approach.

Key-words: Heterogeneous master-worker platform, online scheduling, multiple applications.

This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 —Télécopie +33 4 76 61 52 52

Ordonnancement hors-ligne et en-ligne d'applications concurrentes de types «sacs de tâches» sur plates-formes hétérogènes

Résumé : Les problèmes liés à l'ordonnancement de tâches sont déjà difficiles sur des machines traditionnelles. Ils deviennent encore plus inextricables sur des machines hétérogènes, même lorsque les applications considérées sont facilement parallélisables (de type tâches indépendantes). Nous nous intéressons ici à l'ordonnancement d'applications multiples, sous forme de collections de tâches indépendantes et identiques, sur une plate-forme maître-esclave hétérogène. Les requêtes de calcul surviennent au cours du temps, ce qui signifie que nous ne disposons pas de connaissance sur la charge de travail au tout début de l'exécution. Notre objectif est de minimiser l'étirement (*stretch*) maximum des applications, c'est-à-dire le rapport entre le temps que l'application passe dans le système avant d'être terminée et le temps qu'elle y aurait passé si elle disposait de la plate-forme pour elle seule.

D'un point de vue théorique, nous concevons un algorithme optimal pour le cas hors-ligne (*offline*), lorsque toutes les dates d'arrivée et les caractéristiques des applications sont connues à l'avance. Nous proposons également plusieurs méthodes heuristiques pour le cas en-ligne (*online*), sans connaissance sur l'arrivée future des applications.

D'un point vue expérimental, nous avons mené des expérimentations approfondies sous la forme de simulations avec SimGrid mais aussi dans un environment parallèle réel, en utilisant MPI. Ces expérimentations montrent que nous sommes capables d'ordonnancer des problèmes de grande taille en quelques secondes. Enfin, la solution que nous proposons surpasse les méthodes heuristiques classiques, ce qui démontre l'intérêt de notre démarche.

Mots-clés : Plate-forme maître-esclave hétérogène, ordonnancement en-ligne, applications concurrentes.

1 Introduction

Scheduling problems are already difficult on traditional parallel machines. They become extremely challenging on heterogeneous clusters, even when embarrassingly parallel applications are considered. For instance, consider a bag-of-tasks [1], i.e., an application made of a collection of independent and identical tasks, to be scheduled on a master-worker platform. Although simple, this kind of framework is typical of a large class of problems, including parameter sweep applications [21] and BOINC-like computations [18]. If the master-worker platform is homogeneous, i.e., if all workers have identical CPUs and same communication bandwidths to/from the master, then elementary greedy strategies, such as purely demand-driven approaches, will achieve an optimal throughput. On the contrary, if the platform gathers heterogeneous processors, connected to the master via different-speed links, then the previous strategies are likely to fail dramatically. This is because it is crucial to select which resources to enroll before initiating the computation [5, 41].

In this paper, we still target fully parallel applications, but we introduce a much more complex (and more realistic) framework than scheduling a single application. We envision a situation where users, or clients, submit several bags-of-tasks to a heterogeneous master-worker platform, using a classical client-server model. Applications are submitted online, which means that there is no a priori (static) knowledge of the workload distribution at the beginning of the execution. When several applications are executed simultaneously, they compete for hardware (network and CPU) resources.

What is the scheduling objective in such a framework? A greedy approach would execute the applications sequentially in the order of their arrival, thereby optimizing the execution of each application onto the target platform. Such a simple approach is not likely to be satisfactory for the clients. For example, the greedy approach may delay the execution of the second application for a very long time, while it might have taken only a small fraction of the resources and few time-steps to execute it concurrently with the first one. More strikingly, both applications might have used completely different platform resources (being assigned to different workers) and would have run concurrently at the same speed as in exclusive mode on the platform. Sharing resources to execute several applications concurrently has two key advantages: (i) from the clients' point of view, the average response time (the delay between the arrival of an application and the completion of its last task) is expected to be much smaller; (ii) from the resource usage perspective, different applications will have different characteristics, and are likely to be assigned different resources by the scheduler. Overall, the global utilization of the platform will increase. The traditional measure to quantify the benefits of concurrent scheduling on shared resources is the maximum stretch. The stretch of an application is defined as the ratio of its response time under the concurrent scheduling policy over its response time in dedicated mode, i.e., when it is the only application executed on the platform. The objective is then to minimize the maximum stretch of any application, thereby enforcing a fair trade-off between all applications.

The aim of this paper is to provide a scheduling strategy which minimizes the maximum stretch of several concurrent bags-of-tasks which are submitted online. Our scheduling algorithm relies on complicated mathematical tools but can be computed in time polynomial to

the problem size. On the theoretical side, we prove that our strategy is optimal for the offline version of the problem (when all release dates and application characteristics are known beforehand). We also introduce several heuristics for the general case of online applications. On the practical side, we have conducted extensive simulations and MPI experiments, showing that we are able to deal with very large problem instances in a few seconds. Also, the solution that we compute totally outperforms classical heuristics from the literature, thereby fully assessing the usefulness of our approach.

The rest of the paper is organized as follows. Section 2 describes the platform and application models. Section 3 is devoted to the derivation of the optimal solution in the offline case, and to the presentation of heuristics for online applications. In Section 4 we report an extensive set of simulations and MPI experiments, and we compare the optimal solution against several classical heuristics from the literature. Section 5 is devoted to an overview of related work. Finally, we state some concluding remarks in Section 6.

2 Framework

In this section, we outline the model for the target platforms, as well as the characteristics of the applicative framework. Next we survey steady-state scheduling techniques and we introduce the objective function, namely the maximum stretch of the applications.

2.1 Platform Model

We target a heterogeneous master-worker platform (see Figure 1), also called star network or single-level tree in the literature.

The master P_{master} is located at the root of the tree, and there are p workers P_u ($1 \leq u \leq p$). The link between P_{master} and P_u has a bandwidth b_u. We assume a linear cost model, hence it takes X/b_u time-units to send (resp. receive) a message of size X to (resp. from) P_u. The computational speed of worker P_u is s_u, meaning that it takes X/s_u time-units to execute X floating point operations. Without any loss of generality, we assume that the master has no processing capability. Otherwise, we can simulate the computations of the master by adding an extra worker paying no communication cost.

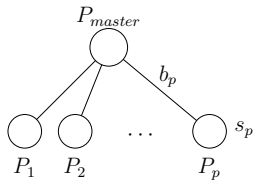


Figure 1: A star network.

2.1.1 Communication models

Traditional scheduling models enforce the rule that computations cannot progress faster than processor speeds would allow: limitations of computation resources are well taken into account. Curiously, these models do not make similar assumptions for communications: in the literature, an arbitrary number of communications may take place at any time-step [50, 20]. In particular, a given processor can send an unlimited number of messages in parallel, and each of these messages is routed as if was alone in the system (no sharing of resources). Obviously, these models are not realistic, and we need to better take communication resources into account. To this purpose, we present two different models, which cover a wide range of practical situations.

Under the *bounded multiport* communication model [33], the master can send/receive data to/from all workers at a given time-step. However, there is a limit on the amount of data that the master can send per time-unit, denoted as BW. In other words, the total amount of data sent by the master to all workers each time-unit cannot exceed BW. Intuitively, the bound BW corresponds to the bandwidth capacity of the master's network card; the flow of data out of the card can be either directed to a single link or split among several links indifferently, hence the multiport hypothesis. The bounded multiport model fully accounts for the heterogeneity of the platform, as each link has a different bandwidth. Simultaneous sends and receives are allowed (all links are assumed bi-directional, or full-duplex).

Another, more restricted model, is the *one-port* model [16, 17]. In this model the master can send data to a single worker at a given time, so that the sending operations have to be serialized. Suppose for example that the master has a message of size X to send to worker P_u. We recall that the bandwidth of the communication link between both processors is b_u. If the transfer starts at time t, then the master cannot start another sending operation before time $t + X/b_u$. Usually, a processor is supposed to be able to perform one send and one receive operation at the same time. However, this hypothesis will not be useful in our study, as the master processor is the only one to send data.

The one-port model seems to fit the performance of some current MPI implementations, which serialize asynchronous MPI sends as soon as message sizes exceed a few hundreds of kilobytes [44]. However, recent multi-threaded communication libraries such as MPICH [32, 34] allow for initiating multiple concurrent send and receive operations, thereby providing practical realizations of the multiport model.

Finally, for both the *bounded multiport* and the *one-port* models, we assume that computation can be overlapped by independent communication, without any interference.

2.1.2 Computation models

We propose two models for the computation. Under the *fluid computation* model, we assume that several tasks can be executed at the same time on a given worker, with a time-sharing mechanism. Furthermore, we assume that we totally control the computation rate for each task. For example, suppose that two tasks A and B are executed on the same worker at respective rates α and β. During a time period Δt, $\alpha \cdot \Delta t$ units of work of task A and $\beta \cdot \Delta t$

units of work of task B are completed. These computation rates may be changed at any time during the computation of a task.

Our second computation model, the *atomic computation* model, assumes that only a single task can be computed on a worker at any given time, and this execution cannot be stopped before its completion (no preemption).

Under both computation models, a worker can only start computing a task once it has completely received the message containing the task. However, for the ease of proofs, we add a variant to the *fluid computation* model, called *synchronous start* computation: in this model, the computation on a worker can start at the same time as the reception of the task starts, provided that the computation rate is smaller than, or equal to, the communication rate (the communication must complete before the computation). This models the fact that, in several applications, only the first bytes of data are needed to start executing a task. In addition, the theoretical results of this paper are more easily expressed under this model, which provides an upper bound on the achievable performance.

2.1.3 Proposed platform model taxonomy

We summarize here the various platform and application models under study:

Bounded Multiport with Fluid Computation and Synchronous Start (BMP-FC-SS). This is the uttermost simple model: communication and computation start at the same time, communication and computation rates can vary over time within the limits of link and processor capabilities. We include this model in our study because it provides a good and intuitive framework to understand the results presented here. This model also provides an upper bound on the achievable performance, which we use as a reference for other models.

Bounded Multiport with Fluid Computation (BMP-FC). This model is a step closer to reality, as it allows computation and communication rates to vary over time, but it imposes that a task input data is completely received before its execution can start.

Bounded Multiport with Atomic Computation (BMP-AC). In this model, two tasks cannot be computed concurrently on a worker. This model takes into account the fact that controlling precisely the computing rate of two concurrent applications is practically challenging, and that it is sometimes impossible to run simultaneously two applications because of memory constraints.

One-Port Model with Atomic Computation (OP-AC). This is the same model as the BMP-AC, but with one-port communication constraint on the master. It represents systems where concurrent sends are not allowed.

In the following, we mainly focus on the variants of the bounded multiport model. We explain the results obtained with the one-port model in Section 3.3.4.

There is a hierarchy among all the multiport models: intuitively, in terms of hardness,

$$\text{BMP-FC-SS} < \text{BMP-FC} < \text{BMP-AC}$$

Formally, a valid schedule for BMP-AC is valid for BMP-FC and a valid schedule for BMP-FC is valid for BMP-FC-SS. This is why studying BMP-FC-SS is useful for deriving upper bounds for all other models.

2.2 Application model

We consider n bags-of-tasks A_k, $1 \leq k \leq n$. The master P_{master} holds the input data of each application A_k upon its release time. Application A_k is composed of a set of $\Pi^{(k)}$ independent, same-size tasks. In order to completely execute an application, all its constitutive tasks must be computed (in any order).

We let $w^{(k)}$ be the amount of computations (expressed in flops) required to process a task of A_k. The speed of a worker P_u may well be different for each application, depending upon the characteristics of the processor and upon the type of computations needed by each application. To take this into account, we refine the platform model and add an extra parameter, using $s_u^{(k)}$ instead of s_u in the following. In other words, we move from the uniform machine model to the unrelated machine model of scheduling theory [20]. The time required to process one task of A_k on processor P_u is thus $w^{(k)}/s_u^{(k)}$. Each task of A_k has a size $\delta^{(k)}$ (expressed in bytes), which means that it takes a time $\delta^{(k)}/b_u$ to send a task of A_k to processor P_u (when there are no other ongoing transfers). For simplicity we do not consider any return message: either we assume that the results of the tasks are stored on the workers, or we merge the return message of the current task with the input message of the next one (and update the communication volume accordingly).

2.3 Steady-state scheduling

Assume for a while that a unique bag-of-tasks A_k is executed on the platform. If $\Pi^{(k)}$, the number of independent tasks composing the application, is large (otherwise, why would we deploy A_k on a parallel platform?), we can relax the problem of minimizing the total execution time. Instead, we aim at maximizing the throughput, i.e., the average (fractional) number of tasks executed per time-unit. We design a cyclic schedule, that reproduces the same schedule every period, except possibly for the very first (initialization) and last (clean-up) periods. It is shown in [9, 5] how to derive an optimal schedule for throughput maximization. The idea is to characterize the optimal throughput as the solution of a linear program over rational numbers, which is a problem with polynomial time complexity.

Throughout the paper, we denote by $\rho_u^{(k)}$ the throughput of worker P_u for application A_k, i.e., the average number of tasks of A_k that P_u executes each time-unit. In the special case where application A_k is executed alone in the platform, we denote by $\rho_u^{*(k)}$ the value of this throughput in the solution which maximizes the total throughput: $\rho^{*(k)} = \sum_{u=1}^{p} \rho_u^{*(k)}$.

We write the following linear program (see Equation (1)), which enables us to compute an asymptotically optimal schedule. The maximization of the throughput is bounded by three types of constraints:

- The first set of constraints state that the processing capacity of P_u is not exceeded.
- The second set of constraints states that the bandwidth of the link from P_{master} to P_u is not exceeded.
- The last constraint states that the total outgoing capacity of the master is not exceeded.

$$(1) \qquad \begin{cases} \text{MAXIMIZE } \rho^{*(k)} = \sum_{u=1}^{p} \rho_u^{*(k)} \text{ SUBJECT TO} \\ \forall 1 \leq u \leq p, \quad \rho_u^{*(k)} \frac{w^{(k)}}{s_u^{(k)}} \leq 1 \\ \forall 1 \leq u \leq p, \quad \rho_u^{*(k)} \frac{\delta^{(k)}}{b_u} \leq 1 \\ \sum_{u=1}^{p} \rho_u^{*(k)} \frac{\delta^{(k)}}{\text{BW}} \leq 1 \end{cases}$$

The formulation in terms of a linear program is simple when considering a single application. In this case, a closed-form expression can be derived. First, the first two sets of constraints can be transformed into:

$$\forall 1 \leq u \leq p \quad \rho_u^{*(k)} \leq \min \left\{ \frac{s_u^{(k)}}{w^{(k)}}, \frac{b_u}{\delta^{(k)}} \right\}.$$

Then, the last constraint can be rewritten:

$$\sum_{u=1}^{p} \rho_u^{*(k)} \leq \frac{\text{BW}}{\delta^{(k)}}.$$

So that the optimal throughput is

$$\rho^{*(k)} = \min \left\{ \frac{\text{BW}}{\delta^{(k)}}, \sum_{u=1}^{p} \min \left\{ \frac{s_u^{(k)}}{w^{(k)}}, \frac{b_u}{\delta^{(k)}} \right\} \right\}.$$

It can be shown [9, 5] that any feasible schedule under one of the multiport model has to enforce the previous constraints. Hence the optimal value $\rho^{*(k)}$ is an upper bound of the achievable throughput. Moreover, we can construct an actual schedule, based on an optimal solution of the linear program and which approaches the optimal throughput. The reconstruction is particularly easy. For example the following procedure builds an asymptotic optimal schedule for the BMP-AC model (bounded multiport communication with atomic computation). As this is the most constrained multiport model, this schedule is feasible in any multiport model:

- While there are tasks to process on the master, send tasks to processor P_u with rate $\rho_u^{*(k)}$.
- As soon as processor P_u starts receiving a task it processes at the rate $\rho_u^{*(k)}$.

Due to the constraints of the linear program, this schedule is always feasible and it is asymptotically optimal, not only among periodic schedules, but more generally among any possible schedules. More precisely, its execution time differs from the minimum execution time by a constant factor, independent of the total number of tasks $\Pi^{(k)}$ to process [5]. This allows us to accurately approximate the total execution time, also called makespan, as:

$$MS^{*(k)} = \frac{\Pi^{(k)}}{\rho^{*(k)}}.$$

We often use $MS^{*(k)}$ as a comparison basis to approximate the makespan of an application when it is alone on the computing platform. If $MS_{\text{opt}}^{(k)}$ is the optimal makespan for this single application, then we have

$$MS_{\text{opt}}^{(k)} - M_k \leq MS^{*(k)} \leq MS_{\text{opt}}^{(k)}$$

where M_k is a fixed constant, independent of $\Pi^{(k)}$.

2.4 Stretch

We come back to the original scenario, where several applications are executed concurrently. Because they compete for resources, their throughput will be lower. Equivalently, their execution rate will be slowed down. Informally, the stretch [12] of an application is the slowdown factor.

Let $r^{(k)}$ be the release date of application A_k on the platform. Its execution will terminate at time $\mathcal{C}^{(k)} \equiv r^{(k)} + MS^{(k)}$, where $MS^{(k)}$ is the time to execute all $\Pi^{(k)}$ tasks of A_k. Because there might be other applications running concurrently to A_k during part or whole of its execution, we expect that $MS^{(k)} \geq MS^{*(k)}$. We define the average throughput $\rho^{(k)}$ achieved by A_k during its (concurrent) execution using the same equation as before:

$$MS^{(k)} = \frac{\Pi^{(k)}}{\rho^{(k)}}.$$

In order to process all applications fairly, we would like to ensure that their actual (concurrent) execution is as close as possible to their execution in dedicated mode. The stretch of application A_k is its slowdown factor

$$\mathcal{S}_k = \frac{MS^{(k)}}{MS^{*(k)}} = \frac{\rho^{*(k)}}{\rho^{(k)}}$$

Our objective function is defined as the *max-stretch* $\mathcal{S}$, which is the maximum of the stretches of all applications:

$$\mathcal{S} = \max_{1 \leq k \leq n} \mathcal{S}_k$$

Minimizing the *max-stretch* $\mathcal{S}$ ensures that the slowdown factor is kept as low as possible for each application, and that none of them is unduly favored by the scheduler.

3 Theoretical study

The main contribution of this paper is a polynomial algorithm to schedule several bag-of-task applications arriving online, while minimizing the maximum stretch. We start this section with the presentation of an asymptotically optimal algorithm for the offline setting, when application release dates and characteristics are known in advance. Then we present our solution for the online framework.

3.1 Offline setting for the fluid model

3.1.1 Defining the set of possible solutions

In this section, we assume that all characteristics of the n applications A_k, $1 \leq k \leq n$ are known in advance.

The scheduling algorithm is the following. Given a candidate value for the max-stretch, we have a procedure to determine whether there exists a solution that can achieve this value. The optimal value will then be found using a binary search on possible values.

Consider a candidate value $\mathcal{S}^l$ for the max-stretch. If this objective is feasible, all applications will have a max-stretch smaller than $\mathcal{S}^l$, hence:

$$\forall\ 1 \leq k \leq n,\ \frac{MS^{(k)}}{MS^{*(k)}} \leq \mathcal{S}^l \iff \forall 1 \leq k \leq n,\ \ \mathcal{C}^{(k)} = r^{(k)} + MS^{(k)} \leq r^{(k)} + \mathcal{S}^l \times MS^{*(k)}$$

Thus, given a candidate value $\mathcal{S}^l$, we have a deadline:

$$d^{(k)} = r^{(k)} + \mathcal{S}^l \times MS^{*(k)} \tag{2}$$

for each application A_k, $1 \leq k \leq n$. This means that the application must complete before this deadline in order to ensure the expected max-stretch. If this is not possible, no solution is found, and a larger max-stretch should be tried by the binary search.

Once a candidate stretch value $\mathcal{S}$ has been chosen, we divide the total execution time into time-intervals whose bounds are epochal times, that is, applications' release dates or deadlines. Epochal times are denoted $t_j \in \{r^{(1)}, ..., r^{(n)}\} \cup \{d^{(1)}, \ldots, d^{(n)}\}$, such that $t_j \leq t_{j+1}$, $1 \leq j \leq 2n-1$. Our algorithm consists in running each application A_k during its whole execution window $[r^{(k)}, d^{(k)}]$, but with a different throughput on each time-interval $[t_j, t_{j+1}]$ such that $r^{(k)} \leq t_j$ and $t_{j+1} \leq d^{(k)}$. Some release dates and deadlines may be equal, leading

to empty time-intervals, for example if there exists j such that $t_j = t_{j+1}$. We do not try to remove these empty time-intervals so as to keep simple indices.

Note that contrarily to the steady-state operation with only one application, in the different time-intervals, the communication throughput may differ from the computation throughput: when the communication rate is larger than the computation rate, extra tasks are stored in a buffer. On the contrary, when the computation rate is larger, tasks are extracted from the buffer and processed. We introduce new notations to take both rates, as well as buffer sizes, into account:

- $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$ denotes the communication throughput from the master to the worker P_u during time-interval $[t_j, t_{j+1}]$ for application A_k, i.e., the average number of tasks of A_k sent to P_u per time-units.
- $\rho^{(k)}_u(t_j, t_{j+1})$ denotes the computation throughput of worker P_u during time-interval $[t_j, t_{j+1}]$ for application A_k, i.e., the average number of tasks of A_k computed by P_u per time-units.
- $B^{(k)}_u(t_j)$ denotes the (fractional) number of tasks of application A_k stored in a buffer on P_u at time t_j.

We write the linear constraints that must be satisfied by the previous variables. Our aim is to find a schedule with minimum stretch satisfying those constraints. Later, based on rates satisfying these constraints, we show how to construct a schedule achieving the corresponding stretch.

All tasks sent by the master. The first set of constraints ensures that all the tasks of a given application A_k are actually sent by the master:

$$\forall\ 1 \le k \le n, \sum_{\substack{1\le j\le 2n-1 \\ t_j\ \ge\ r^{(k)} \\ t_{j+1}\ \le\ d^{(k)}}} \sum_{u=1}^{p} \rho^{(k)}_{M\to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) = \Pi^{(k)}. \tag{3}$$

Non-negative buffers. Each buffer should always have a non-negative size:

$$\forall\ 1 \le k \le n, \forall 1 \le u \le p, \forall 1 \le j \le 2n, \quad B^{(k)}_u(t_j) \ge 0. \tag{4}$$

Buffer initialization. At the beginning of the computation of application A_k, all corresponding buffers are empty:

$$\forall\ 1 \le k \le n, \forall 1 \le u \le p, \quad B^{(k)}_u(r^{(k)}) = 0. \tag{5}$$

Emptying Buffer. After the deadline of application A_k, no tasks of this application should remain on any node:

$$\forall\ 1 \le k \le n, \forall 1 \le u \le p, \quad B^{(k)}_u(d^{(k)}) = 0. \tag{6}$$

Task conservation. During time-interval $[t_j, t_{j+1}]$, some tasks of application A_k are received and some are consumed (computed), which impacts the size of the buffer:

$$\forall\ 1 \le k \le n, \forall 1 \le j \le 2n-1, \forall 1 \le u \le p, \quad B_u^{(k)}(t_{j+1}) = B_u^{(k)}(t_j) + \left(\rho_{M\to u}^{(k)}(t_j, t_{j+1}) - \rho_u^{(k)}(t_j, t_{j+1})\right) \times \left(t_{j+1} - t_j\right) \tag{7}$$

Bounded computing capacity. The computing capacity of a node should not be exceeded on any time-interval:

$$\forall 1 \le j \le 2n-1, \forall 1 \le u \le p, \ \sum_{k=1}^{n} \rho_u^{(k)}(t_j, t_{j+1}) \frac{w^{(k)}}{s_u^{(k)}} \ \le\ 1. \tag{8}$$

Bounded link capacity. The bandwidth of each link should not be exceeded:

$$\forall 1 \le j \le 2n-1, \forall 1 \le u \le p, \ \sum_{k=1}^{n} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{b_u} \ \le\ 1. \tag{9}$$

Limited sending capacity of master. The total outgoing bandwidth of the master should not be exceeded:

$$\forall 1 \le j \le 2n-1, \quad \sum_{u=1}^{p} \sum_{k=1}^{n} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{\mathrm{BW}} \le 1. \tag{10}$$

Non-negative throughputs.

$$\forall 1 \le u \le p, \forall 1 \le k \le n, \forall 1 \le j \le 2n-1, \quad \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \ge 0 \text{ and } \rho_u^{(k)}(t_j, t_{j+1}) \ge 0. \tag{11}$$

We obtain a convex polyhedron (K) defined by the previous constraints. The problem turns now into checking whether the polyhedron is empty and, if not, into finding a point in the polyhedron.

$$(K) \quad \begin{cases} \rho_{M\to u}^{(k)}(t_j, t_{j+1}), \rho_u^{(k)}(t_j, t_{j+1}),\ \forall k, u, j \text{ such that } 1 \le k \le n, 1 \le u \le p, 1 \le j \le 2n-1 \\ \text{under the constraints } (3), (7), (5), (6), (4), (8), (9), (10) \text{ and } (11) \end{cases}$$

3.1.2 Number of tasks processed

At first sight, it may seem surprising that in this set of linear constraints, we do not have an equation establishing that all tasks of a given application are eventually processed. Indeed, such a constraint can be derived from the constraints related to the number of tasks sent from the master and the size of buffers. Consider the constraints on task conservation

(Equation (7)) on a given processor P_u, and for a given application A_k; these equations can be written:

$$\forall 1 \leq j \leq 2n-1, \quad B_u^{(k)}(t_{j+1}) - B_u^{(k)}(t_j) = \left(\rho_{M\to u}^{(k)}(t_j, t_{j+1}) - \rho_u^{(k)}(t_j, t_{j+1})\right) \times \left(t_{j+1} - t_j\right).$$

If we sum all these constraints for all time-interval bounds between $t_{\text{start}} = r^{(k)}$ and $t_{\text{stop}} = d^{(k)}$, we obtain:

$$B_u^{(k)}(t_{\text{stop}}) - B_u^{(k)}(t_{\text{start}}) = \sum_{\substack{[t_j,\ t_{j+1}] \\ t_j \geq r^{(k)} \\ t_{j+1} \leq d^{(k)}}} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \times \left(t_{j+1} - t_j\right) - \sum_{\substack{[t_j,\ t_{j+1}] \\ t_j \geq r^{(k)} \\ t_{j+1} \leq d^{(k)}}} \rho_u^{(k)}(t_j, t_{j+1}) \times \left(t_{j+1} - t_j\right)$$

Thanks to constraints (5) and (6), we know that $B_u^{(k)}(t_{\text{start}}) = 0$ and $B_u^{(k)}(t_{\text{stop}}) = 0$. So the overall number of tasks sent to a processor P_u is equal to the total number of tasks computed:

$$\sum_{\substack{[t_j,\ t_{j+1}] \\ t_j \geq r^{(k)} \\ t_{j+1} \leq d^{(k)}}} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \times \left(t_{j+1} - t_j\right) = \sum_{\substack{[t_j,\ t_{j+1}] \\ t_j \geq r^{(k)} \\ t_{j+1} \leq d^{(k)}}} \rho_u^{(k)}(t_j, t_{j+1}) \times \left(t_{j+1} - t_j\right)$$

This is true for all processors, and constraints (3) tells us that the total number of tasks sent for application A_k is $\Pi^{(k)}$, so:

$$\sum_{u=1}^{p} \sum_{\substack{[t_j,\ t_{j+1}] \\ t_j \geq r^{(k)} \\ t_{j+1} \leq d^{(k)}}} \rho_u^{(k)}(t_j, t_{j+1}) \times \left(t_{j+1} - t_j\right) = \Pi^{(k)}$$

Therefore in any solution in Polyhedron (K), all tasks of each application are processed.

3.1.3 Bounding the buffer size

The size of the buffers could also be bounded by adding constraints:

$$\forall 1 \leq u \leq p, \forall 1 \leq j \leq 2n, \quad \sum_{k=1}^{n} B_u^{(k)}(t_j)\delta^{(k)} \leq M_u$$

where M_u is the size of the memory available on node P_u. We bound the needed memory only at time-interval bounds, but the above argument can be used to prove that the buffer size on P_u never exceeds M_u. We choose not to include this constraint in our basic set of constraints, as this buffer size limitation only applies to the fluid model. Indeed, we have earlier proven that limiting the buffer size for independent tasks scheduling leads to NP-complete problems [10].

3.1.4 Equivalence between non-emptiness of Polyhedron (K) and achievable stretch

Finding a point in Polyhedron (K) allows to determine whether the candidate value for the stretch is feasible. Depending on whether Polyhedron (K) is empty, the binary search will be continued with a larger or smaller stretch value:

- If the polyhedron is not empty, then there exists a schedule achieving stretch $\mathcal{S}$. $\mathcal{S}$ becomes the upper bound of the binary search interval and the search proceeds.
- On the contrary, if the polyhedron is empty, then it is not possible to achieve $\mathcal{S}$. $\mathcal{S}$ becomes the lower bound of the binary search.

This binary search and its proof are described below. For now, we concentrate on proving that the polyhedron is not empty if and only if the stretch $\mathcal{S}$ is achievable.

Note that the previous study assumes a fluid framework, with flexible computing and communicating rates. This is particularly convenient for the totally fluid model (BMP-FC-SS) and we prove below that the algorithm computes the optimal stretch under this model. The strength of our method is that this study is also valid for the other models. The results are slightly different, leading to asymptotic optimality results and the proofs detailed below are slightly more involved. However, this technique allows to approach optimality.

Theorem 1. *Under the totally fluid model, Polyhedron (K) is not empty if and only if there exists a schedule with stretch $\mathcal{S}$.*

In practice, to know if the polyhedron is empty or to obtain a point in (K), we can use classical tools for linear programs, just by adding a fictitious linear objective function to our set of constraints. Some solvers allow the user to limit the number of refinement steps once a point is found in the polyhedron; this could be helpful to reduce the running time of the scheduler.

Proof. $\boxed{\Rightarrow}$ Assume that the polyhedron is not empty, and consider a point in (K), given by the values of the $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$ and $\rho^{(k)}_{u}(t_j, t_{j+1})$. We construct a schedule which obeys exactly these values. During time-interval $[t_j, t_{j+1}]$, the master sends tasks of application A_k to processor P_u with rate $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$, and this processor computes these tasks at a rate $\rho^{(k)}_{u}(t_j, t_{j+1})$.

To prove that this schedule is valid under the fluid model, and that it has the expected stretch, we define $\rho^{(k)}_{M\to u}(t)$ as the instantaneous communication rate, and $\rho^{(k)}_{u}(t)$ as the instantaneous computation rate. Then the (fractional) number of tasks of A_k sent to P_u in interval $[0, T]$ is

$$\int_0^T \rho^{(k)}_{M\to u}(t)dt$$

With the same argument as in the previous remark, applied on interval $[0, T]$, we have

$$B^{(k)}_u(T) = \int_0^T \rho^{(k)}_{M\to u}(t)dt - \int_0^T \rho^{(k)}_{u}(t)dt$$

Since the buffer size is positive for all t_j and evolves linearly in each interval $[t_j, t_{j+1}]$, it is not possible that a buffer has a negative size, so

$$\int_0^T \rho_u^{(k)}(t)dt \leq \int_0^T \rho_{M\to u}^{(k)}(t)dt$$

Hence data is always received before being processed.

With the constraints of Polyhedron (K), it is easy to check that no processor or no link is over-utilized and the outgoing capacity of the master is never exceeded. All the deadlines computed for stretch $\mathcal{S}$ are satisfied by construction, so this schedule achieves stretch $\mathcal{S}$.

$\boxed{\Leftarrow}$ Now we prove that if there exists a schedule S_1 with stretch $\mathcal{S}$, Polyhedron (K) is not empty. We consider such a schedule, and we call $\rho_{M\to u}^{(k)}(t)$ (and $\rho_u^{(k)}(t)$) the communication (and computation) rate in this schedule for tasks of application A_k on processor P_u at time t. We compute as follows the average values for communication and computation rates during time interval $[t_j, t_{j+1}]$:

$$\rho_{M\to u}^{(k)}(t_j, t_{j+1}) = \frac{\displaystyle\int_{t_j}^{t_{j+1}} \rho_{M\to u}^{(k)}(t)dt}{t_{j+1}-t_j} \text{ and } \rho_u^{(k)}(t_j, t_{j+1}) = \frac{\displaystyle\int_{t_j}^{t_{j+1}} \rho_u^{(k)}(t)dt}{t_{j+1}-t_j}.$$

In this schedule, all tasks of application A_k are sent by the master, so

$$\int_{r^{(k)}}^{d^{(k)}} \rho_{M\to u}^{(k)}(t)dt = \Pi^{(k)}.$$

With the previous definitions, Equation (3) is satisfied. Along the same line, we can prove that the task conservation constraints (Equation (7)) are satisfied. Constraints on buffers (Equations 5, 6 and 4) are necessarily satisfied by the size of the buffer in schedule S_1 since it is feasible. Similarly, we can check that the constraints on capacities are verified. □

3.1.5 Binary search

To find the optimal stretch, we perform a binary search using the emptiness of Polyhedron (K) to determine whether it is possible to achieve the current stretch.

The initial upper bound for this binary search is computed using a naive schedule where all applications are computed sequentially. For the sake of simplicity, we consider that all applications are released at time 0 and terminate simultaneously. This is clearly a worst case scenario. We recall that the throughput for a single application on the whole platform can be computed as:

$$\rho^{*(k)} = \min\left\{\frac{\mathrm{BW}}{\delta^{(k)}}, \sum_{u=1}^{p} \min\left\{\frac{s_u^{(k)}}{w^{(k)}}, \frac{b_u}{\delta^{(k)}}\right\}\right\}$$

Then the execution time for application A_k is simply $\Pi^{(k)}/\rho^{*(k)}$. We consider that all applications terminate at time $\sum_k \Pi^{(k)}/\rho^{*(k)}$, so that the worst stretch is

$$\mathcal{S}_{\max} = \max_k \frac{\Pi^{(k)}/\rho^{*(k)}}{\sum_k \Pi^{(k)}/\rho^{*(k)}}.$$

The lower bound on the achievable stretch is 1. Determining the termination criterion of the binary search, that is the minimum gap ϵ between two possible stretches, is quite involved, and not very useful in practice. We focus here on the case where this precision ϵ is given by the user. Please refer to Section 3.4 for a low-complexity technique (a binary search among stretch-intervals) to compute the optimal maximum stretch.

Algorithm 1: Binary search

```
begin
    S_inf ← 1
    S_sup ← S_max
    while S_sup − S_inf > ε do
        S ← (S_sup + S_inf)/2
        if Polyhedron (K) is empty then
            S_inf ← S
        else
            S_sup ← S
    return S_sup
end
```

Suppose that we are given $\epsilon > 0$. The binary search is conducted using Algorithm 1. This algorithm allows us to approach the optimal stretch, as stated by the following theorem.

Theorem 2. *For any $\epsilon > 0$, Algorithm 1 computes a stretch $\mathcal{S}$ such that there exists a schedule achieving $\mathcal{S}$ and $\mathcal{S} \leq \mathcal{S}_{\text{opt}} + \epsilon$, where $\mathcal{S}_{\text{opt}}$ is the optimal stretch. The complexity of Algorithm 1 is $O(\log \frac{\mathcal{S}_{\max}}{\epsilon})$.*

Proof. We prove that at each step, the optimal stretch is contained in the interval $[\mathcal{S}_{\text{inf}}, \mathcal{S}_{\text{sup}}]$ and $\mathcal{S}_{\text{sup}}$ is achievable. This is obvious at the beginning. At each step, we consider the set of constraints for a stretch $\mathcal{S}$ in the interval. If the corresponding polyhedron is empty, Theorem 1 tells us that stretch $\mathcal{S}$ is not achievable, so the optimal stretch is greater than $\mathcal{S}$. If the polyhedron is not empty, there exists a schedule achieving this stretch, thus the optimal stretch is smaller than $\mathcal{S}$.

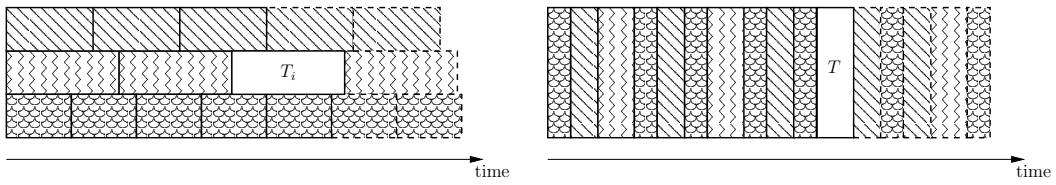
The size of the work interval is divided by 2 at each step, and we stop when this size is smaller than ϵ. Thus the number of steps is $O(\log \frac{\mathcal{S}_{max}}{\epsilon})$. At the end, $\mathcal{S}_{\text{opt}} \in [\mathcal{S}_{\text{inf}}, \mathcal{S}_{\text{sup}}]$ with $\mathcal{S}_{\text{sup}} - \mathcal{S}_{\text{inf}} \leq \epsilon$, so that $\mathcal{S}_{\text{sup}} \leq \mathcal{S}_{\text{opt}} + \epsilon$, and $\mathcal{S}_{\text{sup}}$ is achievable. □

3.2 Property of the one-dimensional load-balancing schedule

Before showing how to extend the previous result to more complex platform models, we introduce a tool that will prove helpful for the proofs: the one-dimensional load-balancing schedule and its properties.

A significant part of this paper is devoted to comparing results under different models. One of the major differences between these models is whether they allow –or not– preemption and time-sharing. On the one hand, we study "fluid" models, where a resource (processor or communication link) can be simultaneously used by several tasks, provided that the total utilization rate is below one. On the other hand, we also study "atomic" models, where a resource can be devoted to only one task, which cannot be preempted: once a task is started on a given resource, this resource cannot perform other tasks before the first one is completed. In this section, we show how to construct a schedule without preemption from fluid schedules, in a way that keeps the interesting properties of the original schedule. Namely, we aim at constructing atomic-model schedules in which tasks terminate not later, or start not earlier, than in the original fluid schedule.

We consider a general case of n applications $A_1, \ldots, A_n$ to be scheduled on the same resource, typically a given processor, and we denote by t_k the time needed to process one task of application A_k at full speed. We start from a fluid schedule S_{fluid} where each application A_k is processed at a rate of α_k tasks per time-units, such that $\sum_{k=1}^{n} \alpha_k \leq 1$. Figure 2(a) illustrates such a schedule.



(a) fluid schedule S_{fluid} (b) atomic schedule S_{1D}

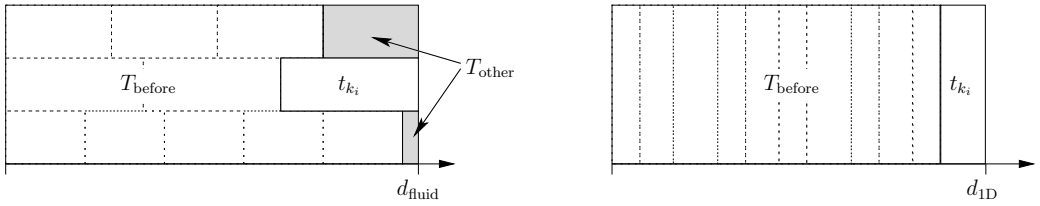
Figure 2: Gantt charts for the proof illustrating the one-dimensional load-balancing algorithm.

From S_{fluid}, we build an atomic-model schedule S_{1D} using a one-dimensional load-balancing algorithm [19, 6]: at any time step, if n_k is the number of tasks of application A_k that have already been scheduled, the next task to be scheduled is the one which minimizes the quantity $\frac{(n_k+1)\times t_k}{\alpha_k}$. Figure 2(b) illustrates the schedule obtained. We now prove that this schedule has the nice property that a task is not processed later in S_{1D} than in S_{fluid}.

Lemma 1. *In the schedule S_{1D}, a task T does not terminate later than in S_{fluid}.*

Proof. First, we point out that t_k/α_k is the time needed to process one task of application A_k in S_{fluid} (with rate α_k). So $\frac{n_k \times t_k}{\alpha_k}$ is the time needed to process the first n_k tasks of application A_k. The scheduling decision which chooses the application minimizing $\frac{(n_k+1)\times t_k}{\alpha_k}$

consists in choosing the task which is not yet scheduled and which terminates first in S_{fluid}. Thus, in $S_{1\text{D}}$, the tasks are executed in the order of their termination date in S_{fluid}. Note that if several tasks terminate at the very same time in S_{fluid}, then these tasks can be executed in any order in $S_{1\text{D}}$, and the partial order of their termination date is still observed in $S_{1\text{D}}$.



Then, consider a task T_i of a given application A_{k_i}, its termination date d_{fluid} in S_{fluid}, and its termination date $d_{1\text{D}}$ in $S_{1\text{D}}$. We call $\mathcal{S}_{\text{before}}$ the set of tasks which are executed before T_i in $S_{1\text{D}}$. Because $S_{1\text{D}}$ executes the tasks in the order of their termination date in S_{fluid}, $\mathcal{S}_{\text{before}}$ is made of tasks which are completed before T_i in S_{fluid}, and possibly some tasks completed at the same time as T_i (at time d_{fluid}). We denote by T_{before} the time needed to process the tasks in $\mathcal{S}_{\text{before}}$.

In $S_{1\text{D}}$, we have $d_{1\text{D}} = T_{\text{before}} + t_{k_i}$ whereas in S_{fluid}, we have $d_{\text{fluid}} = T_{\text{before}} + t_{k_i} + T_{\text{other}}$ where T_{other} is the time spent processing tasks from other application than A_k and which are not completed at time d_{fluid}, or tasks completing at time d_{fluid} and scheduled later than T_i in $S_{1\text{D}}$. Since $T_{\text{other}} \geq 0$, we have $d_{1\text{D}} \leq d_{\text{fluid}}$. □

The previous property is useful when we want to construct an atomic-model schedule, that is a schedule without preemption, in which task results are available no later than in a fluid schedule. On the contrary, it can be useful to ensure that no task will start earlier in an atomic-model schedule than in the original fluid schedule. Here is a procedure to construct a schedule with the latter property.

1. We start again from a fluid schedule S_{fluid}, of makespan M. We transform this schedule into a schedule S_{fluid}^{-1} by reversing the time: a task starting at time d and finishing at time f in S_{fluid} is scheduled to start at time $M - f$ and to terminate at $M - d$ in S_{fluid}^{-1}, and is processed at the same rate as in S_{fluid}. Note that this is possible since we have no precedence constraints between tasks.

2. Then, we apply the previous one-dimensional load-balancing algorithm on S_{fluid}^{-1}, leading to the schedule $S_{1\text{D}}^{-1}$. Thanks to the previous result, we know that a task T does not terminate later in $S_{1\text{D}}^{-1}$ than in S_{fluid}^{-1}.

3. Finally, we transform $S_{1\text{D}}^{-1}$ by reverting the time one last time: we obtain the schedule $S_{1\text{D}}^{-2}$. A task starting at time d and finishing at time f in $S_{1\text{D}}^{-1}$ starts at time $M - f$ and finishes at time $M - d$ in $S_{1\text{D}}^{-2}$. Note that $S_{1\text{D}}^{-1}$ may have a makespan smaller that M (if the resource was not totally used in the original schedule S_{fluid}). In this case, our

method automatically introduces idle time in the one-dimensional schedule, to avoid that a task is started too early.

Lemma 2. *A task does not start sooner in* S_{1D}^{-2} *than in* S_{fluid}.

Proof. Consider a task T, call f_1 its termination date in S_{fluid}^{-1}, and f_2 its termination date in S_{1D}^{-1}. Thanks to Lemma 1, we know that $f_2 \leq f_1$. By construction of the reverted schedules, the starting date of task T in S_{fluid} is $M - f_1$. Similarly, its starting date in S_{1D}^{-2} is $M - f_2$ and we have $M - f_2 \geq M - f_1$. □

3.3 Quasi-optimality for more realistic models

In this section, we explain how the previous optimality result can be adapted to the other models presented in Section 2.1.3. As expected, the more realistic the model, the less tight the optimality guaranty. Fortunately, we are always able to reach *asymptotic optimality*: our schedules get closer to the optimal as the number of tasks per application increases.

We describe the delay induced by each model in comparison to the fluid model: starting from a schedule optimal under the fluid model (BMP-FC-SS), we try to build a schedule with comparable performance under a more constrained scenario.

In the following, we consider a schedule S_1, with stretch $\mathcal{S}$, valid under the totally fluid model (BMP-FC-SS). For the sake of simplicity, we consider that this schedule has been built from a point in Polyhedron (K) as explained in the previous section: the computation and communication rates ($\rho_u^{(k)}(t_j, t_{j+1})$ and $\rho_{M\to u}^{(k)}(t_j, t_{j+1})$) are constant during each interval, and are defined by the coordinates of the point in Polyhedron (K).

We assess the *delay* induced by each model. Given the stretch $\mathcal{S}$, we can compute a deadline $d^{(k)}$ for each application A_k. By moving to more constrained models, we will not be able to ensure that the finishing time $MS^{(k)}$ is smaller than $d^{(k)}$. We call lateness for application A_k the quantity $\max\{0, MS^{(k)} - d^{(k)}\}$, that is the time between the due date of an application and its real termination. Once we have computed the maximum lateness for each model, we show how to obtain asymptotic optimality in Section 3.3.3.

3.3.1 Without simultaneous start: the BMP-FC model

We consider here the BMP-FC model, which differs from the previous model only by the fact that a task cannot start before it has been totally received by a processor.

Theorem 3. *From schedule* S_1*, we can build a schedule* S_2 *obeying the BMP-FC model where the maximum lateness for each application is* $\max_{1\leq u\leq p} \sum_{k=1}^{n} \frac{w^{(k)}}{s_u^{(k)}}$.

Proof. From the schedule S_1, valid under the fluid model (BMP-FC-SS), we aim at building S_2 with a similar stretch where the execution of a task cannot start before the end of the corresponding communication. We first build a schedule as follows, for each processor P_u ($1 \leq u \leq p$):

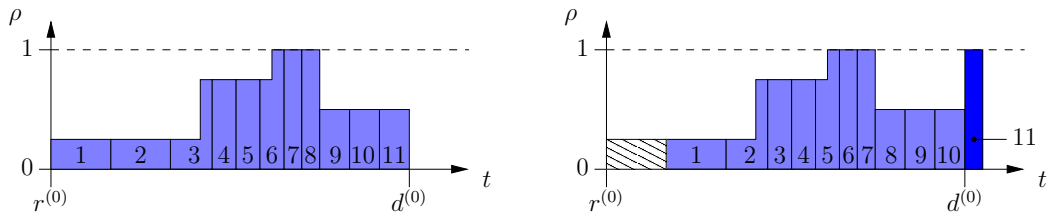
1. Communications to P_u are the same as in S_1;
2. By comparison to S_1, the computations on P_u are shifted for each application A_k: the computation of the first task of A_k is not really performed (P_u is kept idle instead of computing this task), and we replace the computation of task i by the computation of task $i-1$.

Because of the shift of the computations, the last task of application A_k is not executed in this schedule at time $d^{(k)}$. We complete the construction of S_2 by adding some delay after deadline $d^{(k)}$ to process this last task of application A_k at full speed, which takes a time $\frac{w^{(k)}}{s_u^{(k)}}$. All the following computations on processor P_u (in the next time-intervals) are shifted by this delay.

The lateness for any application A_k on processor P_u is at most the sum of the delays for all applications on this processor, $\sum_{k=1}^{n} \frac{w^{(k)}}{s_u^{(k)}}$, and the total lateness of A_k is bounded by the maximum lateness between all processors:

$$lateness^{(k)} \leq \max_{1 \leq u \leq p} \sum_{k=1}^{n} \frac{w^{(k)}}{s_u^{(k)}}$$

An example of such a schedule S_2 is shown on Figure 3 (on a single processor). □



(a) Schedule S_1 (BMP-FC-SS model)

(b) Schedule S_2 (BMP-FC model)

Figure 3: Example of the construction of a schedule S_2 for BMP-FC model from a schedule S_1 for BMP-FC-SS model. We plot only the computing rate. Each box corresponds to the execution of one task.

3.3.2 Atomic execution of tasks: the BMP-AC model

We now move to the BMP-AC model, where a given processor cannot compute several tasks in parallel, and the execution of a task cannot be preempted: a started task must be completed before any other task can be processed.

Theorem 4. *From schedule S_1, we can build a schedule S_3 obeying the BMP-AC model where the maximum lateness for each application is*

$$\max_{1\leq u\leq p} 2n \times \sum_{k=1}^{n} \frac{w^{(k)}}{s_u^{(k)}}.$$

Proof. Starting from a schedule S_1 valid under the fluid model (BMP-FC-SS), we want to build S_3, valid in BMP-AC. We take here advantage of the properties described in Section 3.2 of one-dimensional load-balancing schedules, and especially of $S_{1\mathrm{D}}^{-2}$. Schedule S_3 is built as follows:

1. Communications are kept unchanged;
2. We consider the computations taking place in S_1 on processor P_u during time-interval $[t_j, t_{j+1}]$. A rational number of tasks of each application may be involved in the fluid schedule. We first compute the integer number of tasks of application A_k to be computed in S_3:
$$n_{u,j,k} = \left\lfloor \rho_u^{(k)}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \right\rfloor.$$
The first $n_{u,j,k}$ tasks of A_k scheduled in time-interval $[t_j, t_{j+1}]$ on P_u are organized using the transformation to build $S_{1\mathrm{D}}^{-2}$ in Section 3.2.
3. Then, the computations are shifted as for S_2: for each application A_k, the computation of the first task of A_k is not really performed (the processor is kept idle instead of computing this task), and we replace the computation of task i by the computation of task $i-1$.

Lemma 2 proves that, during time-interval $[t_j, t_{j+1}]$, on processor P_u, a computation does not start earlier in S_3 than in S_1. As S_1 obeys the totally fluid model (BMP-FC-SS), a computation of S_1 does not start earlier than the corresponding communication, so a computation of task i of application A_k in S_1 does not start earlier than the finish time of the communication for task $i-1$ of A_k. Together with the shifting of the computations, this proves that in S_3, the computation of a task does not start earlier than the end of the corresponding communication, on each processor.

Because of the rounding down to the closest integer, on each processor P_u, at each time-interval, S_3 computes at most one task less than S_1 of application A_k. Moreover, one more task computation of application A_k is not performed in S_3 due to the computation shift. On the whole, as there are at most $2n-1$ time-intervals, at most $2n$ tasks of A_k remain to be computed on P_u at time $d^{(k)}$. The delay for application A_k is:

$$\mathit{lateness}^{(k)} \leq \max_{1\leq u\leq p} 2n \times \sum_{k=1}^{n} \times \frac{w^{(k)}}{s_u^{(k)}}.$$

□

This is obviously not the most efficient way to construct a schedule for the BMP-AC model: in particular, each processor is idle during each interval (because of the rounding down). It would certainly be more efficient to sometimes start a task even if it cannot be

terminated before the end of the interval. This is why for our experiments, we implemented on each worker a greedy schedule with Earliest Deadline First Policy instead of this complex construction. However, we can easily prove that this construction has an asymptotic optimal stretch, unlike other greedy strategies.

3.3.3 Asymptotic optimality

In this section, we show that the previous schedules are close to the optimal, when applications are composed of a large number of tasks. To establish such an asymptotic optimality, we have to prove that the gap computed above gets smaller when the number of tasks gets larger. At first sight, we would have to study the limit of the application stretch when $\Pi^{(k)}$ is large for each application. However, if we simply increase the number of tasks in each application without changing the release dates and the tasks characteristics, then the problem will look totally different: any schedule will run for a very long time, and the time separating the release dates will be negligible in front of the whole duration of the schedule. This behavior is not meaningful for our study.

To study the asymptotic behavior of the system, we rather change the granularity of the tasks: we show that when applications are composed of a large number of small-size tasks, then the maximal stretch is close to the optimal one obtained with the fluid model. To take into account the application characteristics, we introduce the granularity g, and we redefine the application characteristics with this new variable:

$$\Pi_g^{(k)} = \frac{\Pi^{(k)}}{g}, \qquad w_g^{(k)} = g \times w^{(k)} \quad \text{and} \quad \delta_g^{(k)} = g \times \delta^{(k)}.$$

When $g = 1$, we get back to the previous case. When $g < 1$, there are more tasks but they have smaller communication and computation size. For any g, the total communication and computation amount per application is kept the same, thus it is meaningful to consider the original release dates.

Our goal is to study the case $g \to 0$. Note that under the totally fluid model (BMP-FC-SS), the granularity has no impact on the performance (or the stretch). Indeed, the fluid model can be seen as the extreme case where $g = 0$. The optimal stretch under the BMP-FC-SS $\mathcal{S}_{\text{opt}}$ does not depend on g.

Theorem 5. *When the granularity is small, the schedule constructed above for the BMP-FC (respectively BMP-AC) model is asymptotically optimal for the maximum stretch, that is*

$$\lim_{g \to 0} \mathcal{S} = \mathcal{S}_{\text{opt}}$$

where $\mathcal{S}$ is the stretch of the BMP-FC (resp. BMP-AC) schedule, and $\mathcal{S}_{\text{opt}}$ the stretch of the optimal fluid schedule.

Proof. The lateness of the applications computed in Section 3.3.1 for the BMP-FC model, and in Section 3.3.2 for the BMP-AC model, becomes smaller when the granularity increase:

for the BMP-FC model, we have

$$lateness^{(k)} \leq \max_{1\leq u\leq p} \sum_{k=1}^{n} \frac{w_g^{(k)}}{s_u^{(k)}} \xrightarrow[g\to 0]{} 0.$$

Similarly, for the BMP-AC model,

$$lateness^{(k)} \leq \max_{1\leq u\leq p} 2n \times \sum_{k=1}^{n} \frac{w_g^{(k)}}{s_u^{(k)}} \xrightarrow[g\to 0]{} 0.$$

Thus, when g gets close to 0, the stretch obtained by these schedules is close to $\mathcal{S}_{\text{opt}}$. □

3.3.4 One-port model

In this section, we explain how to modify the previous study to cope with the one-port model. We cannot simply extend the result obtained for the fluid model to the one-port model (as we have done for the other models) since the parameters for modeling communications are not the same. Actually, the one-port model limits the time spent by a processor (here the master) to send data whereas the multiport model limits its bandwidth capacity. Thus, we have to modify the corresponding constraints. Constraint (10) is replaced by the following one.

(10-b) $$\forall 1 \leq j \leq 2n-1, \sum_{u=1}^{p}\sum_{k=1}^{n} \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{b_u} \leq 1$$

Note that the only difference with Constraint (10) is that, now, we bound the time needed by the master to send all data instead of the volume of the data itself. The set of constraints corresponding to the scheduling problem under the one-port model, for a maximum stretch $\mathcal{S}$, are gathered by the definition of Polyhedron (K_1):

(K_1)
$$\begin{cases} \rho_{M\to u}^{(k)}(t_j, t_{j+1}), \rho_u^{(k)}(t_j, t_{j+1}), \ \forall k, u, j \text{ such that } 1 \leq k \leq n, 1 \leq u \leq p, 1 \leq j \leq 2n-1 \\ \text{under the constraints (3), (7), (5), (6), (4), (8), (9), (10-b), and (11)} \end{cases}$$

As previously, the existence of a point in the polyhedron is linked to the existence of a schedule with stretch $\mathcal{S}$. However, we have no fluid model which could perfectly follow the behavior of the linear constraints. Thus we only target asymptotic optimality.

Theorem 6. *(a) If there exists a schedule valid under the one-port model with stretch $\mathcal{S}_1$, then Polyhedron (K_1) is not empty for $\mathcal{S}_1$.*

(b) Conversely, if Polyhedron (K_1) is not empty for the stretch objective $\mathcal{S}_2$, then there exists a schedule valid for the problem under the one-port model with parameters $\Pi_g^{(k)}$, $\delta_g^{(k)}$, and $w_g^{(k)}$, as defined in Section 3.3.3, whose stretch $\mathcal{S}$ is such that

$$\lim_{g\to 0} \mathcal{S} = \mathcal{S}_2.$$

Proof. (a) To prove the first part of the theorem, we prove that for any schedule with stretch $\mathcal{S}_1$, we can construct a point in Polyhedron (K_1). Given such a schedule, we denote by $A^{(k)}_{M\to u}(t_j, t_{j+1})$ the total number of tasks of application A_k sent by the master to processor P_u during interval $[t_j, t_{j+1}]$. Note that this may be a rational number if there are ongoing transfers at times t_j and/or t_{j+1}. Similarly, we denote by $A^{(k)}_u(t_j, t_{j+1})$ the total (rational) number of tasks of A_k processed by P_u during interval $[t_j, t_{j+1}]$. Then we compute:

$$\rho^{(k)}_{M\to u}(t_j, t_{j+1}) = \frac{A^{(k)}_{M\to u}(t_j, t_{j+1})}{t_{j+1} - t_j} \quad \text{and} \quad \rho^{(k)}_u(t_j, t_{j+1}) = \frac{A^{(k)}_u(t_j, t_{j+1})}{t_{j+1} - t_j}.$$

As in the fluid case, we can also compute the state of the buffers based on these quantities:

$$B^{(k)}_u(t_j) = \sum_{t_i+1\le t_j} A^{(k)}_{M\to u}(t_i, t_{i+1}) - A^{(k)}_u(t_i, t_{i+1})$$

We can easily check that all constraints (3),(4), (5), (6), (7), (8), (9), and (10-b) are satisfied. Variables $B^{(k)}_u(t_j)$, $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$, and $\rho^{(k)}_u(t_j, t_{j+1})$ define a point in Polyhedron (K_1).

(b) From a point in Polyhedron (K_1), we build a schedule which is asymptotically optimal, as defined in Section 3.3.3. During each interval $[t_j, t_{j+1}]$, for each worker P_u, we proceed as follows.

1. We first consider a fluid-model schedule S_f following exactly the rates defined by the point in the polyhedron: the tasks of application A_k are sent with rate $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$ and processed at rate $\rho^{(k)}_u(t_j, t_{j+1})$.
2. We transform both the communication schedule and the computation schedule using one-dimensional load-balancing algorithms. We first compute the integer number of tasks that can be sent in the one-port schedule:

$$n^{\text{comm}}_{u,j,k} = \left\lfloor \rho^{(k)}_{M\to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \right\rfloor.$$

The number of tasks that can be computed on P_u in this time-interval is bounded both by the number of tasks processed in the fluid-model schedule, and by the number of tasks received during this time-interval plus the number of remaining tasks:

$$n^{\text{comp}}_{u,j,k} = \min \left\{ \left\lfloor \rho^{(k)}_u(t_j, t_{j+1}) \times (t_{j+1} - t_j) \right\rfloor , \; n^{\text{comm}}_{u,j,k} + \sum_{i=1}^{j-1} \left(n^{\text{comm}}_{u,i,k} - n^{\text{comp}}_{u,i,k} \right) \right\}$$

The first $n_{u,j,k}^{\text{comm}}$ tasks sent in schedule S_f are organized with the one-dimensional load-balancing algorithm into $S_{1\text{D}}$, while the last $n_{u,j,k}^{\text{comp}}$ tasks executed in schedule S_f are organized with the inverse one-dimensional load-balancing algorithm $S_{1\text{D}}^{-2}$ (see Section 3.2).

3. Then, the computations are shifted: for each application A_k, the computation of the first task of A_k is not really performed (the processor is kept idle instead of computing this task), and we replace the computation of task i by the computation of task $i-1$.

The proof of the validity of the obtained schedule is very similar to the proof of Theorem 4 for the BMP-AC model: we use the fact that a task does not start earlier in $S_{1\text{D}}^{-2}$ than in S_f, and no later in $S_{1\text{D}}$ than in S_f to prove that the data needed for the execution of a given task are received in time.

At time $d^{(k)}$, some tasks of application A_k are still not processed, and some may even not be received yet. Let us denote by L_k the number of time-intervals between $r^{(k)}$ and $d^{(k)}$, that is time-intervals where tasks of application A_k may be processed ($L_k \leq 2n-1$). Because of the rounding of the numbers of tasks sent, at most one task is not transmitted in each interval, for each application. At time $d^{(k)}$, we thus have at most L_k tasks of application A_k to be sent to each processor P_u. We have to serialize the sending operations, which takes a time at most

$$\sum_{u=1}^{n} \frac{L_k \times \delta^{(k)}}{b_u}$$

Then, the number of tasks remaining to be processed on processor P_u is upper bounded by $2L_k+1$: at most L_k are received late because of the rounding of the number of tasks received, at most L_k tasks are received but not computed because we also round the number of tasks processed, and one more task may also remain because of the computation shift. The computation (at full speed) of all these tasks takes at most a time $(2L_k+1)\frac{w^{(k)}}{s_u^{(k)}}$ on processor P_u. Overall, the delay induced on all processors for finishing application A_k can be bounded by:

$$\sum_{u=1}^{n} \frac{L_k \times \delta^{(k)}}{b_u} + \max_{1\leq u\leq p} (2L_k+1) \times \frac{w^{(k)}}{s_u^{(k)}}.$$

As $L_k \leq 2n-1$, the lateness of any application A_k is thus:

$$\mathit{lateness}^{(k)} \leq \sum_k \left(\sum_{u=1}^{n} \frac{(2n-1) \times \delta^{(k)}}{b_u} + \max_{1\leq u\leq p} (4n-1) \times \frac{w^{(k)}}{s_u^{(k)}} \right).$$

As in the proof of Theorem 5, when the granularity becomes small, the stretch of the obtained schedule becomes as close to $\mathcal{S}_2$ as we want. □

3.4 Binary search with stretch-intervals

In this section, we present another method to compute the optimal stretch in the offline case. This method is based on a linear program built from the constraints of the convex polyhedron (K) with the minimization of the stretch as objective. To do this, we need that other parameters (especially the deadlines) are functions of the stretch. We recall that the deadlines of the applications are computed from their release date and the targeted stretch $\mathcal{S}$:

$$d^{(k)} = r^{(k)} + \mathcal{S} \times MS^{*(k)}.$$

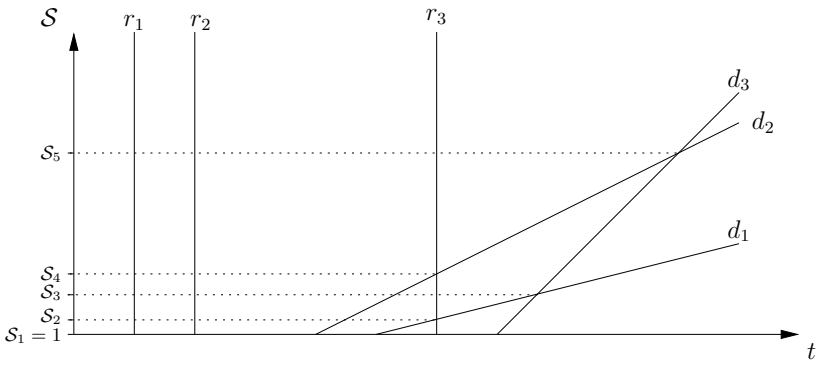


Figure 4: Relation between stretch and deadlines

Figure 4 represents the evolution of the deadlines $d^{(k)}$ over the targeted stretch $\mathcal{S}$: each deadline is an affine function in $\mathcal{S}$. For the sake of readability, the time is represented on the x axis, and the stretch on the y axis. Special values of stretches $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m$ are represented on the figure. These *critical values* of the stretch are points where the ordering of the release dates and deadlines of the applications is modified:

- When $\mathcal{S}$ is such a *critical value*, some release dates and deadlines have the same values;
- When $\mathcal{S}$ varies between two such critical values, i.e., when $\mathcal{S}_a < \mathcal{S} < \mathcal{S}_{a+1}$, then the ordering of the release dates and the deadlines is preserved.

To simplify our notations, we add two artificial *critical values* corresponding to the natural bound of the stretch: $\mathcal{S}_1 = 1$ and $\mathcal{S}_m = \infty$.

Our goal is to find the optimal stretch by slicing the stretch space into a number of intervals. Within each interval defined by the *critical values*, the deadlines are linear functions of the stretch. We first show how to find the best stretch within a given interval using a single linear program, and then how to explore the set of intervals with a binary search, so as to find the one containing the optimal stretch.

3.4.1 Within a stretch-interval

In the following, we work on one stretch-interval, called $[\mathcal{S}_a, \mathcal{S}_b]$. For all values of $\mathcal{S}$ in this interval, the release dates $r^{(k)}$ and deadlines $d^{(k)}$ are in a given order, independent of the value of $\mathcal{S}$. As previously, we note $\{t_j\}_{j=1\ldots 2n} = \{r^{(k)}, d^{(k)}\}$, with $t_j \leq t_{j+1}$. As the values of the t_j may change when $\mathcal{S}$ varies, we write $t_j = \alpha_j \mathcal{S} + \beta_j$. This notation is general enough for all $r^{(k)}$ and $d^{(k)}$:

- If $t_j = r^{(k)}$, then $\alpha_j = 0$ and $\beta_j = r^{(k)}$.
- If $t_j = d^{(k)}$, then $\alpha_j = MS^{*(k)}$ and $\beta_j = r^{(k)}$.

Note that like previously, some t_j might be equal, and especially when the stretch reaches a bound of the stretch-interval ($\mathcal{S} = \mathcal{S}_a$ or $\mathcal{S} = \mathcal{S}_b$), that is a critical value. For the sake of simplicity, we do not try to discard the empty time-intervals, to avoid the renumbering of the epochal times.

When we rewrite the constraints defining the convex polyhedron (K) with these new notations, we obtain quadratic constraints instead of linear constraints. To avoid this, we introduce new notations. Instead of considering the instantaneous communication and computation rates, we use the total amount of tasks sent or computed during a given time-interval. Formally we define $A^{(k)}_{M\to u}(t_j, t_{j+1})$ to be the fractional number of tasks of application A_k sent by the master to processor P_u during the time-interval $[t_j, t_{j+1}]$. Similarly, we denote by $A^{(k)}_u(t_j, t_{j+1})$ the fractional number of tasks of application A_k computed by processor P_u during the time-interval $[t_j, t_{j+1}]$. Of course, these quantities are linked to our previous variables. Indeed, we have:

$$\begin{aligned} A^{(k)}_{M\to u}(t_j, t_{j+1}) &= \rho^{(k)}_{M\to u}(t_j, t_{j+1}) \times (t_{j+1} - t_j) \\ A^{(k)}_u(t_j, t_{j+1}) &= \rho^{(k)}_u(t_j, t_{j+1}) \times (t_{j+1} - t_j) \end{aligned}$$

with $t_{j+1} - t_j = (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)$.

We rewrite the set of constraints with these new notations:

Total number of tasks We make sure that all tasks of application A_k are sent by the master:

$$\forall\ 1 \leq k \leq n, \quad \sum_{\substack{1\leq j\leq 2n-1 \\ t_j\ \geq\ r^{(k)} \\ t_{j+1}\ \leq\ d^{(k)}}} \sum_{u=1}^{p} A^{(k)}_{M\to u}(t_j, t_{j+1}) = \Pi^{(k)} \tag{12}$$

Non-negative buffer Each buffer should always have a non-negative size:

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p, \forall 1 \leq j \leq 2n, \quad B^{(k)}_u(t_j) \geq 0 \tag{13}$$

Buffer initialization At the beginning of the computation of application A_k, all corresponding buffers are empty:

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p, \quad \text{for } t_j = r^{(k)}, \quad B_u^{(k)}(t_j) = 0 \tag{14}$$

Emptying Buffer After the deadline of application A_k, no tasks of this application should remain on any node:

$$\forall\ 1 \leq k \leq n, \forall 1 \leq u \leq p, \quad \text{for } t_j = d^{(k)}, \quad B_u^{(k)}(t_j) = 0 \tag{15}$$

Task conservation During time-interval $[t_j, t_{j+1}]$, some tasks of application A_k are received and some are consumed (computed), which impacts the size of the buffer:

$$\forall\ 1 \leq k \leq n, \forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p, \quad B_u^{(k)}(t_{j+1}) = B_u^{(k)}(t_j) + A_{M\to u}^{(k)}(t_j, t_{j+1}) - A_u^{(k)}(t_j, t_{j+1}) \tag{16}$$

Bounded computing capacity The computing capacity of a node should not be exceeded on any time-interval:

$$\forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p, \sum_{k=1}^{n} A_u^{(k)}(t_j, t_{j+1}) \frac{w^{(k)}}{s_u^{(k)}} \leq (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j) \tag{17}$$

Bounded link capacity The bandwidth of each link should not be exceeded:

$$\forall 1 \leq j \leq 2n-1, \forall 1 \leq u \leq p, \sum_{k=1}^{n} A_{M\to u}^{(k)}(t_j, t_{j+1}) \frac{\delta^{(k)}}{b_u} \leq (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j) \tag{18}$$

Limited sending capacity of master The total outgoing bandwidth of the master should not be exceeded:

$$\forall 1 \leq j \leq 2n-1, \sum_{u=1}^{p} \sum_{k=1}^{n} A_{M\to u}^{(k)}(t_j, t_{j+1}) \delta^{(k)} \leq \mathrm{BW} \times \big((\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)\big) \tag{19}$$

We also add a constraint to bound the objective stretch to be in the targeted stretch-interval:

$$\mathcal{S}_a \leq \mathcal{S} \leq \mathcal{S}_b \tag{20}$$

Even if the bounds of the sum on the time-intervals in Equation (12) seem to depend on $\mathcal{S}$, the set of intervals involved in the sum does not vary as the order of the t_j values is fixed for $\mathcal{S}_a \leq \mathcal{S} \leq \mathcal{S}_b$. With the objective of minimizing the stretch, we get the following linear program.

$$\text{(LP)} \begin{cases} \text{Minimize } \mathcal{S}, \\ \text{under the constraints (12), (13), (14), (15), (16), (17), (18), (19), (20)} \end{cases}$$

Solving this linear program allows to find the minimum possible stretch in the stretch-interval $[\mathcal{S}_a, \mathcal{S}_b]$. If the minimum stretch computed by the linear program is $\mathcal{S}_{\text{opt}} > \mathcal{S}_a$, this means that there is not better possible stretch in $[\mathcal{S}_a, \mathcal{S}_b]$, and thus there is no better stretch for all possible values. On the contrary, if $\mathcal{S}_{\text{opt}} = \mathcal{S}_a$, we cannot conclude: $\mathcal{S}_a$ may be the optimal stretch, or the optimal stretch is smaller than $\mathcal{S}_a$. In this case, the binary search is continued with smaller stretch values. At last, if there is no solution to the linear program, then there exists no possible stretch smaller or equal to $\mathcal{S}_b$, and the binary search is continued with larger stretch values. This binary search and its proof are described below.

When $\mathcal{S}_a < \mathcal{S}_{\text{opt}} \leq \mathcal{S}_b$, we can prove that $\mathcal{S}_{\text{opt}}$ is the optimal stretch.

Theorem 7. *The linear program* (LP) *finds the optimal stretch provided that the optimal stretch is in* $]\mathcal{S}_a, \mathcal{S}_b]$.

Proof. The proof highly depends on Theorem 1. First, consider an optimal solution of the linear program (LP). We compute

$$\rho^{(k)}_{M\to u}(t_j, t_{j+1}) = \frac{A^{(k)}_{M\to u}(t_j, t_{j+1})}{(\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)} \quad \text{and} \quad \rho^{(k)}_{u}(t_j, t_{j+1}) = \frac{A^{(k)}_{u}(t_j, t_{j+1})}{(\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)}.$$

These variables constitute a valid solution of the set of constraints of Theorem 1 for $\mathcal{S} = \mathcal{S}_{\text{opt}}$. Therefore there exists a schedule achieving stretch $\mathcal{S}_{\text{opt}}$.

Assume now that there exists a schedule with stretch $\mathcal{S}$ such that $\mathcal{S}_a < \mathcal{S} < \mathcal{S}_b$. Due to Theorem 1, there exists values for $\rho^{(k)}_{M\to u}(t_j, t_{j+1})$ and $\rho^{(k)}_{u}(t_j, t_{j+1})$ satisfying the corresponding set of constraints for $\mathcal{S}$. Then we compute

$$\begin{aligned} A^{(k)}_{M\to u}(t_j, t_{j+1}) &= \rho^{(k)}_{M\to u}(t_j, t_{j+1}) \times \big((\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)\big) \\ A^{(k)}_{u}(t_j, t_{j+1}) &= \rho^{(k)}_{u}(t_j, t_{j+1}) \times \big((\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)\big) \end{aligned}$$

$A^{(k)}_{M\to u}(t_j, t_{j+1})$ and $A^{(k)}_{u}(t_j, t_{j+1})$ constitute a solution of the linear program (LP) with objective value $\mathcal{S}$. As the objective value $\mathcal{S}_{\text{opt}}$ found by the linear program is minimal among all possible solutions, we have $\mathcal{S}_{\text{opt}} \leq \mathcal{S}$. □

3.4.2 Binary search among stretch intervals

We assume that we have computed the bounds of the stretch intervals: $\mathcal{S}_1, \ldots, \mathcal{S}_m$. The binary search to reach the optimal stretch works as follows:

Theorem 8. *Algorithm 2 finds the optimal stretch value in a polynomial number of steps.*

Algorithm 2: Binary search among stretch-intervals

```
begin
    L ← 1 and U ← max
    while U − L > 1 do
        M ← ⌊(L + U)/2⌋
        Solve the linear program (LP) for interval [S_M, S_{M+1}]
        if there is a solution with objective value S_opt then
            if S_opt > S_M then
                return S_opt
            else
                U ← M
        else
            L ← M
    Solve the linear program (LP) for interval [S_L, S_U]
    return the objective value S_opt of the solution
end
```

Proof. This algorithm performs a binary search among the m stretch-intervals. Thus, the number of steps of this search is $O(\log m)$ and each step consists in solving a linear program, which can be done in polynomial time.

We prove that the optimal stretch is always contained in the interval $[\mathcal{S}_L, \mathcal{S}_U]$. This is obviously true in the beginning. On a stretch-interval $[\mathcal{S}_M, \mathcal{S}_{M+1}]$, the minimum possible stretch $\mathcal{S}_{\text{opt}}$ is computed. If $\mathcal{S}_{\text{opt}} > \mathcal{S}_M$, thanks to Theorem 7, we know that $\mathcal{S}_{\text{opt}}$ is the optimal stretch. If there is no solution, no stretch values in the stretch-interval $[\mathcal{S}_M, \mathcal{S}_{M+1}]$ is feasible, so the optimal stretch is in $[\mathcal{S}_{M+1}, \mathcal{S}_U]$. If $\mathcal{S}_{\text{opt}} = \mathcal{S}_M$, then the optimal stretch smaller or equal than $\mathcal{S}_M$. Thus, the optimal stretch is still contained in $[\mathcal{S}_M, \mathcal{S}_{M+1}]$ after one iteration. If we exit *while* loop without having return the optimal stretch, then $U = L+1$ and the optimal stretch is contained in the stretch-interval $[\mathcal{S}_L, \mathcal{S}_U]$. We compute this value with the linear program and return it. □

3.5 Online setting

We now move to the study of the online setting. Because we target an online framework, the scheduling policy needs to be modified upon the completion of an application, or upon the arrival of a new one. Resources will be re-assigned to the various applications in order to optimize the objective function. The scheduler is making best use of its partial knowledge of the whole process (we know neither the release date, nor the number of tasks, nor the characteristics of the next application to arrive into the system). The idea is to make use of our study of the offline case. When a new application is released, we recompute the achievable max-stretch using the binary search described in the offline case. However, we

cannot pretend to optimality any longer as we now have only limited information on the applications.

When a new application $A_{k_{\text{new}}}$ arrives at time $T_{\text{new}} = r^{(k_{\text{new}})}$, we consider the applications $A_0, \ldots, A_{k_{\text{new}}-1}$, released before T_{new}.

We call $\Pi_{\text{rem}}^{(k)}$ the (fractional) number of tasks of application A_k remaining at the master at time T_{new}. For the sake of simplicity, we do not consider the applications that are totally processed, and we thus have $\Pi_{\text{rem}}^{(k)} \neq 0$ for all applications. For the new application, we have $\Pi_{\text{rem}}^{(k_{\text{new}})} = \Pi^{(k_{\text{new}})}$. We also consider as parameters the state $B_u^{(k)}(t_{k_{\text{new}}})$ of the buffers at time T_{new}. We also have $B_u^{(k_{\text{new}})}(t_{k_{\text{new}}}) = 0$

As previously, we compute the optimal max-stretch using Algorithm 1. For a given objective $\mathcal{S}$, we have a convex polyhedron defined by the linear constraints, which is non empty if and only if stretch $\mathcal{S}$ is achievable. The constraints are slightly modified in order to fit the online context. First, we recompute the deadlines of the applications: $d^{(k)} = r^{(k)} + \mathcal{S} \times MS^{*(k)}$. Note that now, all release dates are smaller than T_{new}, and all deadlines are larger than T_{new}.

We sort the deadlines by increasing order, and denote by t_j the set of orderer deadlines: $\{t_j\} = \{d^{(k)}\} \cup \{T_{\text{new}}\}$ such that $t_j \leq t_{j+1}$. The constraints are the same as the ones used for Polyhedron (K), except the constraint on the number of task processed, which is updated to account for the remaining number of tasks to be processed.

As described for the offline setting, a binary search allows to find the optimal max-stretch. Note that this "optimality" concerns only the time interval $[T_{\text{new}}, +\infty]$, assuming that no other application will be released after T_{new}. This assumption will not hold true in general, hence our schedule will be suboptimal (which is the price to pay without information about future released applications). The stretch achieved for the whole application set is bounded by the maximum of the stretches obtained by the binary search each time a new application is released.

4 MPI experiments and SimGrid simulations

We have conducted several experiments in order to compare different scheduling strategies, and to show the benefits of the algorithms presented in this work. We first present the heuristics. Then we detail the platforms and applications used for the experiments. Finally, we expose and comment the numerical results.

The code and the experimental results can be downloaded from:
http://graal.ens-lyon.fr/~lmarchal/cbs3m/.

4.1 Heuristics

In this section, we present strategies that are able to schedule multi-applications in an online setting. Most of these strategies are simple and wait for an application to terminate before

scheduling another application. Although far from the optimal in a number of cases, such strategies are representative of existing Grid schedulers.

We compare sixteen algorithms in the experiments. First we outline policies for selecting the set of applications to be executed:

FIFO (First In First Out): applications are computed in the order of their release dates.

SPT (Shortest Processing Time): released applications are sorted by non-decreasing processing time (which is approximated by MS^*, see Section 2.3). The first application must be completed before we determine the next one to be executed.

SRPT (Shortest Remaining Processing Time): at each release date, released applications are sorted by non-decreasing processing time, according to the tasks that remains to be scheduled, and the applications are fully executed one after the other in this order until a new release date occurs.

SWRPT (Shortest Weighted Remaining Processing Time): it is very similar to **SRPT**, but the remaining processing time of the released applications are weighted with MS^*. In practice, it gives small applications a priority against large applications which are almost finished, which is better in order to minimize the stretch.

The importance and relevance of the above heuristics are outlined in the related work section (Section 5). Next we outline policies for resource selection:

RR (Round-Robin): all workers are selected in a cyclic way.

MCT (Minimum Completion Time): given a task of an application, it selects the worker which will finish this task first, given the current load of the platform.

DD (Demand-Driven): workers are themselves asking for a task to compute as soon as they become idle.

The four application selection policies and the three resource selection rules lead to twelve different greedy algorithms. We also test a more sophisticated algorithm:

MWMA (Master Worker Multi-Applications): this algorithm computes on each time interval a steady-state strategy to schedule the available applications, as presented in [7, 8]. All available applications are running at the same time, and each application is given a different fraction of the platform according to its weight. This weight can be derived from:

- the remaining number of tasks of the applications (variant called **NBT**);
- the remaining time of computation of the applications (variant called **MS**).

Both variants are compared in the experiments.

Finally, there is the strategy presented in this paper, called **CBS3M** (Clever Burst Steady-State Stretch Minimization). We test it with two variants, both a FIFO or EDF policy for the workers to choose the next task to compute among those they have received. Both the **CBS3M** and the **MWMA** strategies make use of linear programs to compute their schedule. These linear programs are solved using `glpk`, the Gnu Linear Programming Kit [30].

4.2 Platforms

In this section, we conduct experiments on a real platform, in order to have an insight of the behavior of the algorithms. We also run multiple simulations, in order to get more results about their performance. For the sake of simplicity, we suppose in this section that the processors are related, which means that the computation time of a task of each set will only depends on the size of the task and of the computation speed of the worker, and not on the applications.

Because our MPI library serializes communications, we use the one-port model for all experiments and simulations: the linear program used for **CBS3M** is the one adapted for the one-port model (see Section 3.3.4), and we serialize communications both in the MPI program and in the simulations.

4.2.1 Experimental settings

Experiments were conducted on a cluster composed of nine processors. The master is a SuperMicro server 6013PI, with a P4 Xeon 2.4 GHz processor, and the workers are all SuperMicro servers 5013-GM, with P4 2.4 GHz processors. All nodes have 1 GB of memory and are running Linux. They are connected with a switched 10 Mbps Fast Ethernet network. As this platform may not be as heterogeneous as we would like, we sometimes artificially enhance its heterogeneity by slowing down some communications and/or some computations. In order to artificially slow down a communication link, we send several times the same message to one worker. The same idea works for processor speeds: we ask a worker to compute a given matrix-product several times in order to slow down its computation capability. The experiments are performed using the MPICH-2 communication library [31].

We create ten different fully heterogeneous platforms. The communication and computation slowdowns were uniformly chosen between 1 to 10.

4.2.2 Simulations

An extensive set of simulations is performed using SimGrid [36]. The parameters of the simulated platforms were kept as close as possible to the actual experimental framework so that simulations can be considered a direct complement of the experimental MPI setting. In a first step, we run the exact same experiments (with the same platform configuration and application scenario) to make sure that our simulation behaves similarly to the MPI experiments. Then, we conducted an extensive set of simulations with larger applications.

4.3 Applications

A bag-of-tasks is described by its release date, its number of tasks, and the communication and computation sizes of one task. For our experiments and simulations, we randomly generated the applications, with the following constraints in order to be realistic:

1. the release dates of the applications follow a log-normal distribution as suggested in [27];
2. the total amount of communications and computations for an application is randomly chosen with a log-normal distribution between realistic bounds, and then split into tasks. The parameters used in the generation of the applications for the experiments and the simulations are described in Tables 1 and 2.

general	number of workers	8
	number of applications	12
arrival dates	mean of the distribution in the log space	4.0
	standard deviation in the log space	1.2
computations	maximum amount of work application	76.8 Gflops
	minimum amount of work per task	3.1 Gflops
communications	maximum amount of communication per application	800 MB
	minimum amount of communication per task	40 MB
number of tasks	minimum number of tasks per application	10

Table 1: Parameters for the MPI experiments

general	number of workers	10
	number of applications	20
arrival dates	mean of the distribution in the log space	4.0
	standard deviation in the log space	1.2
computations	maximum amount of work application	409 Gflops
	minimum amount of work per task	3.1 Gflops
communications	maximum amount of communication per application	6 GB
	minimum amount of communication per task	40 MB
number of tasks	minimum number of tasks per application	10

Table 2: Parameters for the SimGrid simulations

The number of tasks for one application is bounded above by the minimum amount of communication and computation allowed for one task.

4.4 Results

In this section we describe the results obtained on all different platforms, experimental or simulated.

4.4.1 Experimental results

We express the performance of any given algorithm on one problem instance as the ratio of the max-stretch obtained by the algorithm on this instance over the theoretical optimal max-stretch obtained by linear programming.

The results of the experiments are shown in Table 3. Figure 5 summarizes the experiments for the best four algorithms, **CBS3M** using EDF policy, in both the offline and online versions, **MWMA NBT** and **SWRPT**.

Algorithm	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7	Exp8	Exp9	Exp10	Average
CBS3M EDF OFFLINE	**1.20**	**1.21**	**1.27**	**1.32**	**1.18**	1.16	1.34	1.68	1.28	1.13	1.28
CBS3M EDF ONLINE	1.28	1.25	1.35	1.45	1.37	**1.14**	**1.27**	1.45	1.45	**1.09**	1.31
CBS3M FIFO OFFLINE	1.38	1.25	1.28	1.37	1.34	1.22	1.35	1.64	**1.27**	1.37	1.35
CBS3M FIFO ONLINE	1.42	1.26	1.48	1.43	1.47	1.15	1.54	1.55	1.36	1.16	1.38
FIFO MCT	1.71	2.46	1.87	2.54	1.53	1.28	2.77	1.66	2.27	1.37	1.95
FIFO RR	5.06	3.03	2.88	3.58	4.31	4.42	3.75	9.37	3.70	2.55	4.26
MWMA MS	1.66	1.99	2.42	1.80	2.17	2.18	1.80	2.98	2.28	3.18	2.24
MWMA NBT	1.22	1.45	1.43	1.53	1.53	1.63	1.36	1.67	1.48	1.49	1.48
SPT DD	4.27	3.06	2.36	2.74	5.00	9.20	4.18	11.17	3.33	2.32	4.76
SPT MCT	1.89	2.48	1.71	1.99	2.17	1.74	2.78	**1.28**	2.30	1.37	1.97
SRPT MCT	1.91	2.41	1.72	2.00	2.17	1.76	2.79	1.64	2.27	1.38	2.00
SWRPT MCT	1.92	2.44	1.72	1.99	2.17	1.76	2.97	1.63	2.28	1.38	2.03

Table 3: Results of the experiments.

We can see in Table 3 that the four versions of **CBS3M** achieve a better relative max-stretch than most other strategies. In fact, they all achieve far better performance than any other strategy in all but two experiments. We also see that resource selection is important on heterogeneous platforms, as the algorithms which have the worst relative max-stretch are the ones using round-robin or demand-driven policies. The **MWMA** algorithms lie in between our algorithms and the greedy strategies, but sometimes they achieve a very bad relative max-stretch (up to 2.98).

On Figure 5, one can clearly see that our algorithms outperform the other algorithms. Surprisingly, the offline version is not always better than the online version. The offline version knows the future and thus should achieve better performance. However, it suffers from discrepancies between the actual characteristics of the platform and those of the platform model. The online version is able to circumvent this problem as it takes into account the work effectively processed to recompute the schedule at each new application arrival. This gain of reactivity compensates for the loss due to the lack of knowledge of the future.

4.4.2 Simulations on experimental platforms

We have run exactly the same experiments with simulations, using the same platform configurations and application scenarios. We compare the difference between the relative max-stretch in both cases. As a result, we found that the average difference on the relative

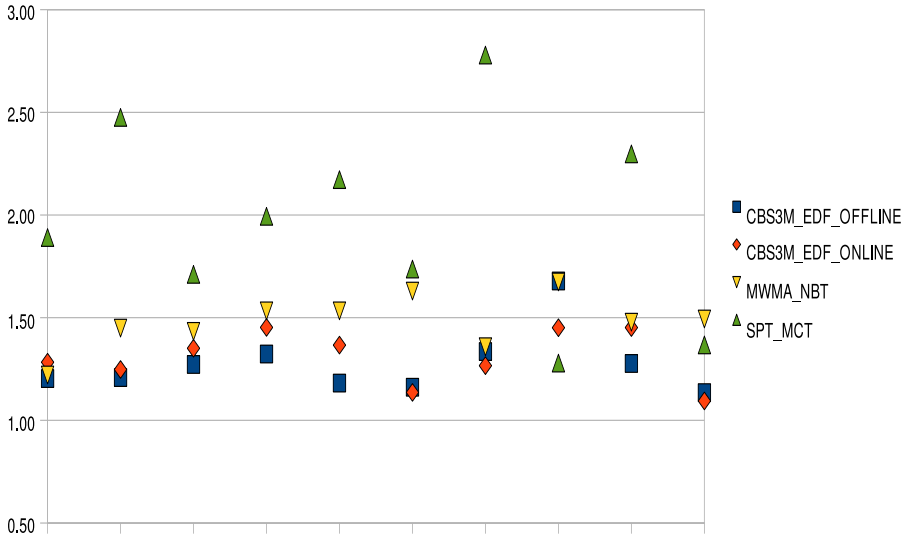


Figure 5: Relative max-stretch of best four heuristics.

max-stretch (that is the ratio between the max-stretch obtain by any heuristic and the optimal max-stretch computed by the linear program) is around 21%, with a standard deviation of 57%. These results show that our simulations are generally close to those obtained on a real platform. Indeed, only one scenario has very different executions in the MPI experiment and in the simulation, with a 566% difference. In this case, the slowdown of the processors is not correctly achieved, leading for the **SPT** scheduler to take a totally different decision. If we discard this execution, the average difference drops down to 16%, with a standard deviation of 14% (and a maximum of 72%).

4.4.3 Simulation results

In this section, we detail the results of the simulations. We run 1000 experiments based on the parameters described in Table 2. Table 4 presents the results of all heuristics for the max-stretch metric, whereas Figure 6 shows the evolution of some heuristics (the best ones) over the load of the scenario. Here the load is characterized with the optimal achievable max-stretch: we consider that a scenario where the optimal max-stretch is 6 is twice as loaded as a scenario with an optimal max-stretch of 3.

The **CBS3M** heuristics perform very well for the max-stretch: **CBS3M** EDF ONLINE achieves the optimal max-stretch in 65.2% of the experiments. This heuristic achieves great performance, with an average max-stretch of 1.16 times the optimal max-stretch, and a worst case of 1.93.

More surprisingly, **CBS3M** also gives the best average results for the makespan and the max-flow objectives. With respect to sum-flow, **CBS3M** gives the best results for light-loaded scenarios, whereas **SRPT** and **SWRPT** give better results for high-loaded scenarios. Finally, **CBS3M** is outperformed by **SRPT** and **SWRPT** for sum-stretch.

The good results of the **CBS3M** heuristics can be explained by the fact that they make very good use of the platform, by scheduling simultaneously several applications when it is possible, for example when the communication medium has still some free bandwidth after scheduling the most critical application. All other heuristics (except **MWMA**) are limited to scheduling only one application at a time, leading to an overall bad utilization of the computing platform.

Another comment is the relative bad result of the involved strategies **MWMA** (**MWMA NBT** and **MWMA MS**): although they schedule several applications concurrently on the platforms, they use a somewhat wrong computation of the priorities, leading to poor results.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	4.550	16.689	(± 7.897)	62.6	(the best in 0.0 %)
FIFO_MCT	1.857	6.912	(± 2.404)	17.9	(the best in 0.0 %)
FIFO_DD	4.550	16.689	(± 7.897)	62.6	(the best in 0.0 %)
SPT_RR	1.348	4.274	(± 1.771)	13.8	(the best in 0.0 %)
SPT_MCT	1.007	1.928	(± 0.610)	5.99	(the best in 1.3 %)
SPT_DD	1.348	4.274	(± 1.771)	13.8	(the best in 0.0 %)
SRPT_RR	1.348	4.121	(± 1.737)	13.8	(the best in 0.0 %)
SRPT_MCT	1.007	1.861	(± 0.601)	6.87	(the best in 2.2 %)
SRPT_DD	1.348	4.121	(± 1.737)	13.8	(the best in 0.0 %)
SWRPT_RR	1.344	4.119	(± 1.739)	13.8	(the best in 0.0 %)
SWRPT_MCT	1.007	1.857	(± 0.601)	6.87	(the best in 1.9 %)
SWRPT_DD	1.344	4.119	(± 1.739)	13.8	(the best in 0.0 %)
MWMA_NBT	1.477	3.433	(± 1.044)	8.49	(the best in 0.0 %)
MWMA_MS	2.435	8.619	(± 2.420)	20.4	(the best in 0.0 %)
CBS3M_FIFO_ONLINE	1.003	1.322	(± 0.208)	2.83	(the best in 6.9 %)
CBS3M_EDF_ONLINE	**1.003**	**1.163**	**(± 0.118)**	**1.93**	**(the best in 64.0 %)**
CBS3M_FIFO_OFFLINE	1.022	1.379	(± 0.276)	3.74	(the best in 3.8 %)
CBS3M_EDF_OFFLINE	**1.011**	**1.213**	**(± 0.125)**	**2.06**	**(the best in 26.2 %)**

Table 4: Max-stretch of all heuristics in the simulations.

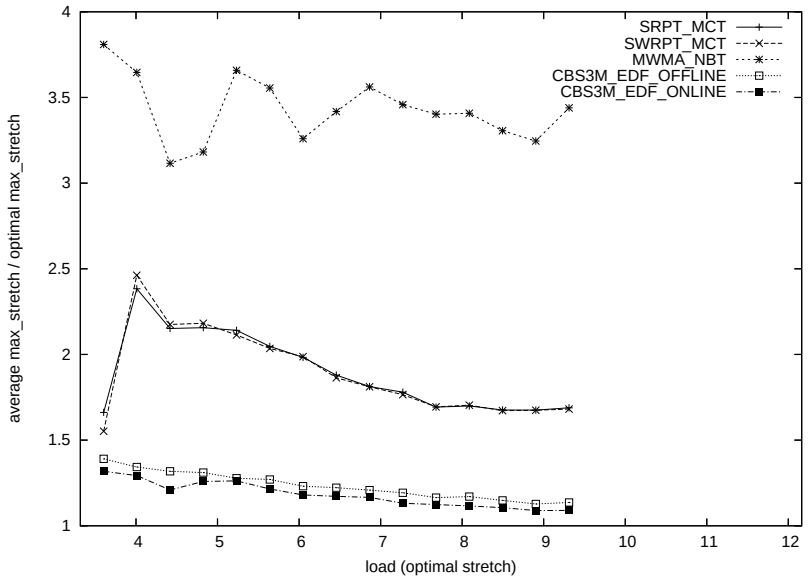


Figure 6: Evolution of the relative max-stretch of best heuristics in the simulations under different load conditions.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	2.064	6.783	(± 3.210)	30.7	(the best in 0.0 %)
FIFO_MCT	1.322	2.754	(± 0.670)	6.45	(the best in 0.0 %)
FIFO_DD	2.064	6.783	(± 3.210)	30.7	(the best in 0.0 %)
SPT_RR	1.019	2.942	(± 1.221)	10.1	(the best in 0.0 %)
SPT_MCT	1.000	1.182	(± 0.183)	2.53	(the best in 2.4 %)
SPT_DD	1.019	2.942	(± 1.221)	10.1	(the best in 0.0 %)
SRPT_RR	1.007	2.607	(± 1.071)	8.93	(the best in 0.0 %)
SRPT_MCT	**1.000**	**1.045**	**(± 0.098)**	**1.92**	**(the best in 25.5 %)**
SRPT_DD	1.007	2.607	(± 1.071)	8.93	(the best in 0.0 %)
SWRPT_RR	1.000	2.596	(± 1.068)	8.96	(the best in 0.1 %)
SWRPT_MCT	**1.000**	**1.038**	**(± 0.098)**	**1.92**	**(the best in 60.1 %)**
SWRPT_DD	1.000	2.596	(± 1.068)	8.96	(the best in 0.1 %)
MWMA_NBT	1.051	2.013	(± 0.644)	5.41	(the best in 0.0 %)
MWMA_MS	1.663	4.183	(± 1.269)	11.5	(the best in 0.0 %)
CBS3M_FIFO_ONLINE	1.000	1.294	(± 0.208)	2.16	(the best in 0.4 %)
CBS3M_EDF_ONLINE	**1.000**	**1.201**	**(± 0.190)**	**2.08**	**(the best in 20.2 %)**
CBS3M_FIFO_OFFLINE	1.000	1.332	(± 0.227)	2.57	(the best in 0.1 %)
CBS3M_EDF_OFFLINE	1.000	1.272	(± 0.214)	2.49	(the best in 3.8 %)

Table 5: Sum-stretch of all heuristics in the simulations.

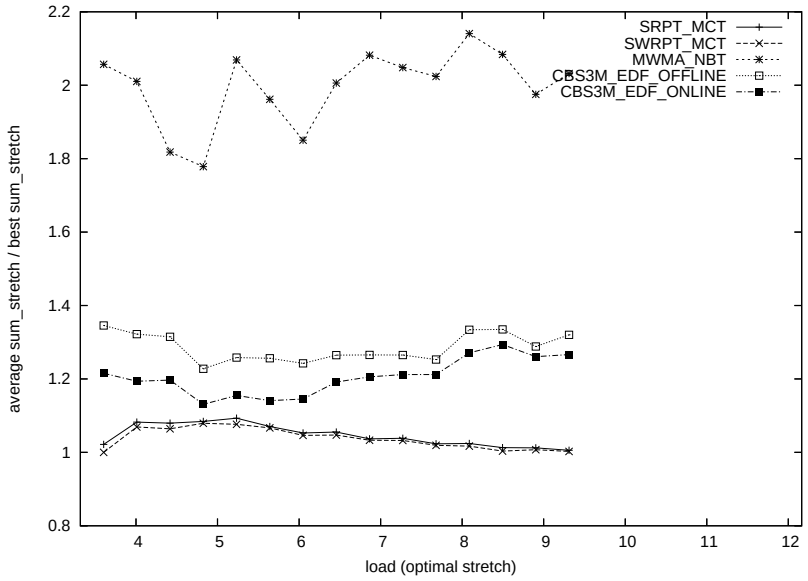


Figure 7: Evolution of the sum-stretch of best heuristics in the simulations under different load conditions.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	1.343	2.716	(± 0.684)	5.31	(the best in 0.0 %)
FIFO_MCT	1.000	1.329	(± 0.202)	2.11	(the best in 0.1 %)
FIFO_DD	1.343	2.716	(± 0.684)	5.31	(the best in 0.0 %)
SPT_RR	1.325	2.714	(± 0.685)	5.33	(the best in 0.0 %)
SPT_MCT	1.000	1.329	(± 0.202)	2.1	(the best in 0.0 %)
SPT_DD	1.325	2.714	(± 0.685)	5.33	(the best in 0.0 %)
SRPT_RR	1.325	2.714	(± 0.686)	5.32	(the best in 0.0 %)
SRPT_MCT	1.000	1.328	(± 0.202)	2.1	(the best in 0.0 %)
SRPT_DD	1.325	2.714	(± 0.686)	5.32	(the best in 0.0 %)
SWRPT_RR	1.322	2.715	(± 0.686)	5.32	(the best in 0.0 %)
SWRPT_MCT	1.000	1.328	(± 0.202)	2.1	(the best in 0.0 %)
SWRPT_DD	1.322	2.715	(± 0.686)	5.32	(the best in 0.0 %)
MWMA_NBT	1.000	1.079	(± 0.070)	1.45	(the best in 4.6 %)
MWMA_MS	1.000	1.078	(± 0.067)	1.42	(the best in 2.1 %)
CBS3M_FIFO_ONLINE	1.000	1.029	(± 0.029)	1.17	(the best in 7.5 %)
CBS3M_EDF_ONLINE	**1.000**	**1.004**	**(± 0.006)**	**1.05**	**(the best in 35.0 %)**
CBS3M_FIFO_OFFLINE	**1.000**	**1.018**	**(± 0.023)**	**1.22**	**(the best in 17.6 %)**
CBS3M_EDF_OFFLINE	**1.000**	**1.003**	**(± 0.006)**	**1.07**	**(the best in 53.0 %)**

Table 6: Makespan of all heuristics in the simulations.

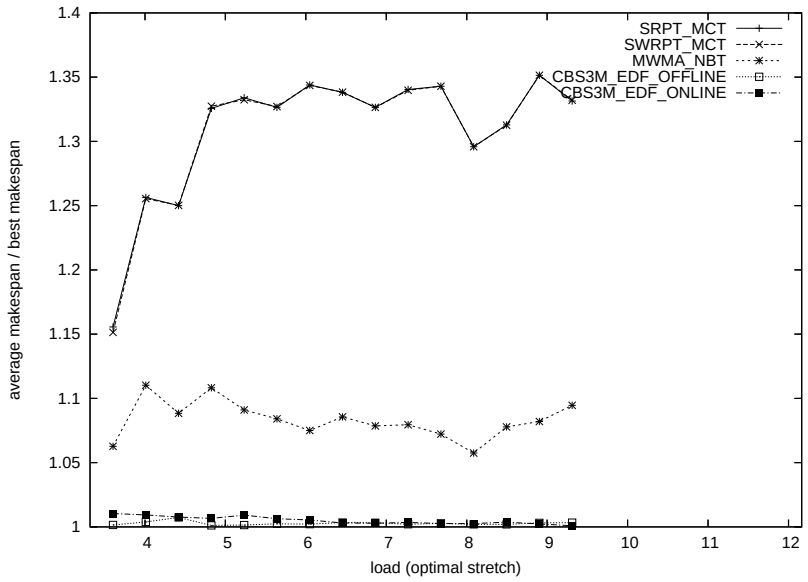


Figure 8: Evolution of the makespan of best heuristics in the simulations under different load conditions.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	1.146	3.097	(± 1.135)	10.2	(the best in 0.0 %)
FIFO_MCT	**1.000**	**1.281**	**(± 0.258)**	**2.83**	**(the best in 14.4 %)**
FIFO_DD	1.146	3.097	(± 1.135)	10.2	(the best in 0.0 %)
SPT_RR	1.386	3.282	(± 1.222)	10.9	(the best in 0.0 %)
SPT_MCT	1.002	1.460	(± 0.287)	3.09	(the best in 0.0 %)
SPT_DD	1.386	3.282	(± 1.222)	10.9	(the best in 0.0 %)
SRPT_RR	1.386	3.289	(± 1.225)	10.9	(the best in 0.0 %)
SRPT_MCT	1.003	1.473	(± 0.306)	4.28	(the best in 0.0 %)
SRPT_DD	1.386	3.289	(± 1.225)	10.9	(the best in 0.0 %)
SWRPT_RR	1.382	3.291	(± 1.225)	10.9	(the best in 0.0 %)
SWRPT_MCT	1.000	1.477	(± 0.309)	4.28	(the best in 0.1 %)
SWRPT_DD	1.382	3.291	(± 1.225)	10.9	(the best in 0.0 %)
MWMA_NBT	1.000	1.181	(± 0.153)	1.99	(the best in 7.0 %)
MWMA_MS	1.000	1.261	(± 0.189)	2.32	(the best in 1.1 %)
CBS3M_FIFO_ONLINE	1.000	1.054	(± 0.061)	1.52	(the best in 5.8 %)
CBS3M_EDF_ONLINE	**1.000**	**1.031**	**(± 0.057)**	**1.48**	**(the best in 23.2 %)**
CBS3M_FIFO_OFFLINE	**1.000**	**1.037**	**(± 0.058)**	**1.48**	**(the best in 21.6 %)**
CBS3M_EDF_OFFLINE	**1.000**	**1.023**	**(± 0.055)**	**1.48**	**(the best in 48.7 %)**

Table 7: Max-flow of all heuristics in the simulations.

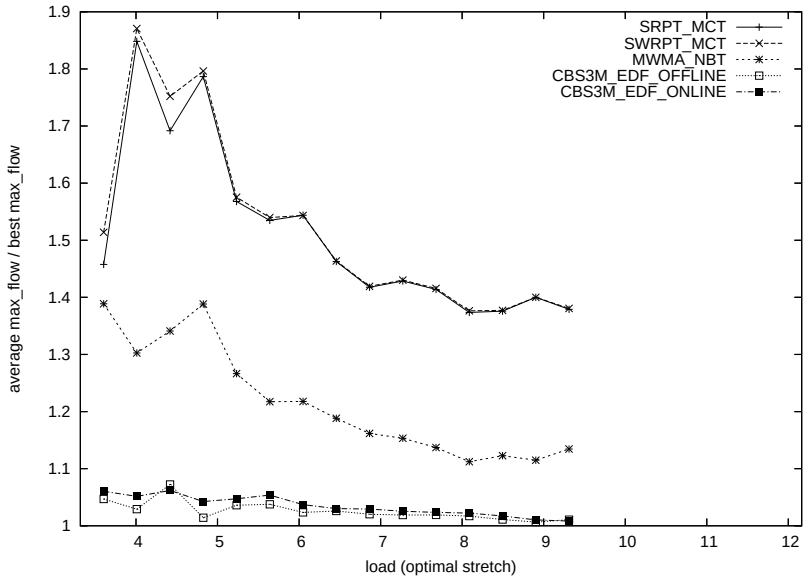


Figure 9: Evolution of the max-flow of best heuristics in the simulations under different load conditions.

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
FIFO_RR	1.644	4.020	(± 1.567)	16.3	(the best in 0.0 %)
FIFO_MCT	1.134	1.652	(± 0.264)	3.33	(the best in 0.0 %)
FIFO_DD	1.644	4.020	(± 1.567)	16.3	(the best in 0.0 %)
SPT_RR	1.196	2.811	(± 1.081)	9.21	(the best in 0.0 %)
SPT_MCT	1.000	1.149	(± 0.171)	2.32	(the best in 3.5 %)
SPT_DD	1.196	2.811	(± 1.081)	9.21	(the best in 0.0 %)
SRPT_RR	1.079	2.704	(± 1.048)	9.03	(the best in 0.0 %)
SRPT_MCT	**1.000**	**1.105**	**(± 0.151)**	**2.23**	**(the best in 32.1 %)**
SRPT_DD	1.079	2.704	(± 1.048)	9.03	(the best in 0.0 %)
SWRPT_RR	1.079	2.706	(± 1.049)	9.03	(the best in 0.0 %)
SWRPT_MCT	**1.000**	**1.108**	**(± 0.152)**	**2.23**	**(the best in 15.4 %)**
SWRPT_DD	1.079	2.706	(± 1.049)	9.03	(the best in 0.0 %)
MWMA_NBT	1.000	1.404	(± 0.217)	2.29	(the best in 0.1 %)
MWMA_MS	1.359	2.333	(± 0.355)	3.7	(the best in 0.0 %)
CBS3M_FIFO_ONLINE	1.000	1.122	(± 0.101)	1.62	(the best in 1.4 %)
CBS3M_EDF_ONLINE	**1.000**	**1.065**	**(± 0.090)**	**1.53**	**(the best in 35.6 %)**
CBS3M_FIFO_OFFLINE	1.000	1.120	(± 0.103)	1.67	(the best in 0.3 %)
CBS3M_EDF_OFFLINE	**1.000**	**1.087**	**(± 0.101)**	**1.66**	**(the best in 18.7 %)**

Table 8: Sum-flow of all heuristics in the simulations.

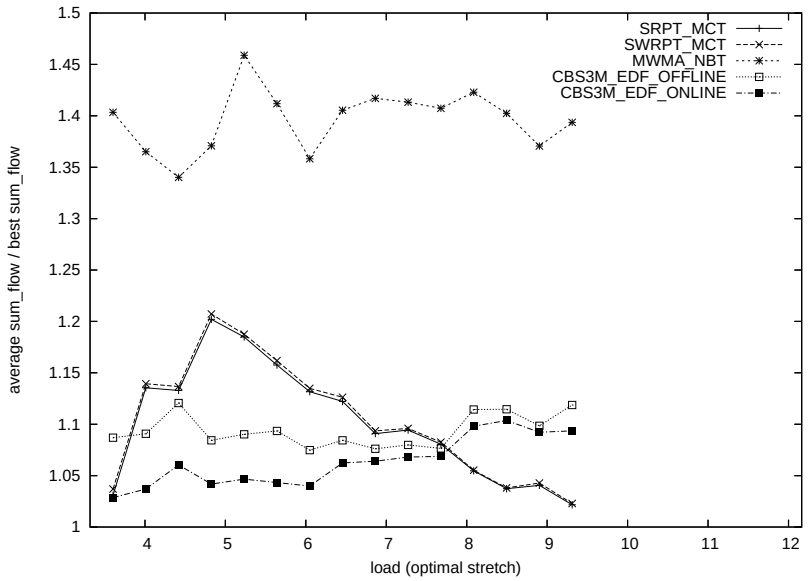


Figure 10: Evolution of the sum-flow of best heuristics in the simulations under different load conditions.

5 Related work

Related literature can be classified into three main categories: (i) bags-of-tasks; (ii) steady-state scheduling; and (iii) flow-type objective functions and online scheduling.

5.1 Bags-of-Tasks

Bags-of-tasks are parallel applications whose tasks are all independent. Their study is motivated by problems that are addressed by collaborative computing efforts such as SETI@home [46], factoring large numbers [25], the Mersenne prime search [42], and those distributed computing problems organized by companies such as Entropia [26]. Bags-of-tasks are well suited for computational grids, because communication can easily become a bottleneck for tightly-coupled parallel applications.

Condor [39] and APST [14, 21] are among the first projects providing specific support for such applications. Condor was initially conceived for campus-wide networks [39], but has been extended to run on grids [28]. While APST is user-centric and does not handle multiple-applications, Condor is system-centric. Those two projects are designed for standard grids but more recent and active projects like OurGrid [24] or BOINC [18] target more distributed architectures like desktop grids. BOINC [18] is a centralized scheduler that distributes tasks for participating applications, such as SETI@home, ClimatePrediction.NET, and Einstein@Home. The set of resources is thus very large while the set of applications is small and very controlled. OurGrid is a Brazilian project that encourages people to donate their computing resources while maintaining the symmetry between consumers and providers. All these projects generally focus on designing and providing a working infrastructure, and they do not provide any analysis of scheduling techniques suited to such environments.

5.2 Steady-State Scheduling

Minimizing the makespan, i.e., the total execution time, is a NP-hard problem in most practical situations [29, 47, 23], while it turns out that the optimal steady-state schedule can often be characterized very efficiently, with low-degree polynomial complexity.

The steady-state approach has been pioneered by Bertsimas and Gamarnik [15]. It has been used successfully in many situations [11]. In particular, steady-state scheduling has been used to schedule independent tasks on heterogeneous tree-overlay networks [9, 5]. Bandwidth-centric scheduling is introduced in [9], and extensive experiments are reported in [35]. The steady-state approach has also been used by Hong et al. [33] who extend the work of [9] to deploy a divisible workload on a heterogeneous platform. However, and to the best of our knowledge, the only reference dealing with steady-state scheduling for several applications is [7].

5.3 Flow-type objective functions and online scheduling

The flow of a task is the time it spends in the system, that is the time elapsed between its release date and its completion time. The stretch of a task is therefore a weighted form of its flow time, where the weight is the inverse of the task running time, if it were alone on the platform. Most of the existing work on stretch minimization deals with the mono-processor case. In fact, there has been a lot of work on the performance of simple list scheduling heuristics for the optimization of flow-like metrics with preemption. We will therefore first consider this work.

Flow optimization. On a single processor, the max-flow is optimized by *First-Come First-Serve* (FCFS) (see Bender et al. [12] for example), and the sum-flow is optimized by *shortest remaining processing time first* (SRPT) [4].

Things are more difficult for stretch minimization. First, any online algorithm which has a better competitive ratio for sum-stretch minimization than FCFS is subject to starvation, and is thus not a competitive algorithm for max-stretch minimization [38]. In other words, the two objective functions cannot be optimized simultaneously to obtain a non trivial competitive factor (FCFS is not taking into account the weight of tasks in the objective).

Sum-stretch minimization. The complexity of the offline minimization of the sum-stretch with preemption is still an open problem. At the very least, this is a hint at the difficulty of this problem. Bender, Muthukrishnan, and Rajaraman [13] designed a Polynomial Time Approximation Scheme (PTAS) for minimizing the sum-stretch with preemption. Chekuri and Khanna [22] proposed an approximation scheme for the more general sum weighted flow minimization problem. On the online side, no online algorithm has a competitive ratio less than or equal to 1.19484 for the minimization of sum-stretch [37, 38].

As we recalled, on one processor, SRPT is optimal for minimizing the sum-flow. When SRPT takes a scheduling decision, it only considers the *remaining* processing time of a task, and not its *original* processing time, i.e., the *weight* of the task in the objective function. Nevertheless, Muthukrishnan, Rajaraman, Shaheen, and Gehrke have shown [40] that SRPT is 2-competitive for sum-stretch. Another well studied algorithm is the Smith's ratio rule [49] also known as *shortest weighted processing time* (SWPT). Whatever the weights, SWPT is 2-competitive [45] for the minimization of the sum of weighted completion times. However, SWPT is not an approximation algorithm for minimizing the sum-stretch. Indeed, both SPT (*shortest processing time*) and SWPT are not competitive algorithms for minimizing the sum-stretch [37, 38]. To address the weaknesses of both SRPT and SWPT, one might consider a heuristic that takes into account both the original and the remaining processing times of the jobs, which leads to the *shortest weighted remaining processing time* heuristic (SWRPT). Muthukrishnan, Rajaraman, Shaheen, and Gehrke [40] proved that SWRPT is actually optimal when there are only two job sizes. However, in the general case, the worst case for SWRPT for sum-stretch minimization is at least 2, and thus is no better than that of SRPT [37, 38].

Max-stretch minimization. Max-stretch can be optimally minimized in the offline case [37, 38], even on unrelated machines (either with preemption or in the divisible load framework). The online case is far more difficult. With only two task sizes, SWRPT is optimal, as we have already recalled. However, as soon as there are at least three task sizes, no algorithm as a competitive ration lower than $\frac{1}{2}\Delta^{\sqrt{2}-1}$, where Δ is the ratio of the largest to the smallest size of tasks [37, 38].

In fact, this latter work is the only one targeting max stretch minimization in a multiprocessor environment. This work is done in the divisible load framework, meaning that applications can be arbitrarily divided in sub-tasks when, in the context of the current paper, the granularity of the tasks of each application is fixed independently of the scheduler. Furthermore, communications can be neglected for the applications targeted in [37, 38], when they play a major role in our case.

General online scheduling. More generally, we refer the reader to surveys on online scheduling algorithms [43], on randomized online scheduling algorithms [2], or even more generally on online algorithms [3].

6 Conclusion

In this paper, we have studied the problem of scheduling multiple applications, made of collections of independent and identical tasks, on a heterogeneous master-worker platform. Applications have different release dates. We aimed at minimizing the maximum stretch, or equivalently at minimizing the largest relative slowdown of each application due to their concurrent execution. We derived an optimal algorithm for the off-line setting (when all application sizes and release dates are known beforehand). We have adapted this algorithm to an online scenario, so that it can react when new applications are released.

We have compared our new algorithms against classical greedy heuristics, and also against some involved static multi-applications strategies. Experiments were run both on a real cluster, using MPI, and through extensive simulations, conducted with SimGrid. Both experimental comparisons show a great improvement when using our **CBS3M** strategy, which achieves an averaged worse max-stretch only 16% greater than the off-line optimal max-stretch. To the best of our knowledge, this work is the first attempt to provide efficient scheduling techniques for multiple bags-of-tasks in an online scenario.

Future work includes extending the approach to other communication models (such as the one-port model of [48]) and to more general platforms (such as multi-level trees). It would also be very interesting to deal with more complex application types, such as pipeline or even general DAGs.

References

[1] M. Adler, Y. Gong, and A. L. Rosenberg. Optimal sharing of bags of tasks in heterogeneous clusters. In *15th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA'03)*, pages 1–10. ACM Press, 2003.

[2] S. Albers. On randomized online scheduling. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 134 – 143. ACM, 2002.

[3] S. Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

[4] K. Baker. *Introduction to Sequencing and Scheduling.* Wiley, New York, 1974.

[5] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Trans. Parallel Distributed Systems*, 15(4):319–330, 2004.

[6] O. Beaumont, V. Boudet, A. Petitet, F. Rastello, and Y. Robert. A proposal for a heterogeneous cluster ScaLAPACK (dense linear solvers). *IEEE Trans. Computers*, 50(10):1052–1070, 2001.

[7] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert. Centralized versus distributed schedulers for multiple bag-of-task applications. In *International Parallel and Distributed Processing Symposium IPDPS'2006.* IEEE Computer Society Press, 2006.

[8] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, L. Marchal, and Y. Robert. Centralized versus distributed schedulers for multiple bag-of-task applications. *IEEE Trans. Parallel Distributed Systems*, 19, 2008, to appear.

[9] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium (IPDPS'2002).* IEEE Computer Society Press, 2002.

[10] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Independent and divisible tasks scheduling on heterogeneous star-schaped platforms with limited memory. In *PDP'2005, 13th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, pages 179–186. IEEE Computer Society Press, 2005.

[11] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Steady-state scheduling on heterogeneous clusters. *Int. J. of Foundations of Computer Science*, 16(2):163–194, 2005.

[12] M. A. Bender, S. Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th Annual ACM-SIAM Symposium On Discrete Algorithms (SODA'98)*, pages 270–279. Society for Industrial and Applied Mathematics, 1998.

[13] M. A. Bender, S. Muthukrishnan, and R. Rajaraman. Approximation algorithms for average stretch scheduling. *J. of Scheduling*, 7(3):195–222, 2004.

[14] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the grid using AppLeS. *IEEE Trans. Parallel Distributed Systems*, 14(4):369–382, 2003.

[15] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithms for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[16] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. In *ICDCS'99 19th International Conference on Distributed Computing Systems*, pages 15–24. IEEE Computer Society Press, 1999.

[17] P. Bhat, C. Raghavendra, and V. Prasanna. Efficient collective communication in distributed heterogeneous systems. *Journal of Parallel and Distributed Computing*, 63(3):251–263, 2003.

[18] BOINC: Berkeley Open Infrastructure for Network Computing. http://boinc.berkeley.edu.

[19] P. Boulet, J. Dongarra, Y. Robert, and F. Vivien. Static tiling for heterogeneous computing platforms. *Parallel Computing*, 25:547–568, 1999.

[20] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, 2004.

[21] H. Casanova and F. Berman. Grid'2002. In F. Berman, G. Fox, and T. Hey, editors, *Parameter sweeps on the grid with APST*. Wiley, 2002.

[22] C. Chekuri and S. Khanna. Approximation schemes for preemptive weighted flow time. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 297–305. ACM Press, 2002.

[23] P. Chrétienne, E. G. Coffman Jr., J. K. Lenstra, and Z. Liu, editors. *Scheduling Theory and its Applications*. John Wiley and Sons, 1995.

[24] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvé, F. A. B. da Silva, C. O. Barros, and C. Silveira. Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. In *Proceedings of the ICCP'2003 - International Conference on Parallel Processing*, Oct. 2003.

[25] J. Cowie, B. Dodson, R.-M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, and J. Zayer. A world wide number field sieve factoring record: on to 512 bits. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology - Asiacrypt '96*, volume 1163 of *LNCS*, pages 382–394. Springer Verlag, 1996.

[26] Entropia. URL: http://www.entropia.com.

[27] D. G. Feitelson. *Workload Characterization and Modeling Book*. electronic draft, no published yet.

[28] J. Frey, T. Tannenbaum, I. Foster, M. Livny, , and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*. IEEE Computer Society Press, Aug. 2001.

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

[30] GLPK: GNU Linear Programming Kit. http://www.gnu.org/software/glpk/.

[31] W. Gropp. Mpich2: A new start for mpi implementations. In *PVM/MPI*, page 7, 2002.

[32] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996. see also http://www-unix.mcs.anl.gov/mpi/mpich/.

[33] B. Hong and V. Prasanna. Distributed adaptive task allocation in heterogeneous computing environments to maximize throughput. In *International Parallel and Distributed Processing Symposium IPDPS'2004*. IEEE Computer Society Press, 2004.

[34] N. T. Karonis, B. Toonen, and I. Foster. Mpich-g2: A grid-enabled implementation of the message passing interface. *J.Parallel and Distributed Computing*, 63(5):551–563, 2003.

[35] B. Kreaseck. *Dynamic autonomous scheduling on Heterogeneous Systems*. PhD thesis, University of California, San Diego, 2003.

[36] A. Legrand, L.Marchal, and H. Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, pages 138–145, May 2003.

[37] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of biological requests. In *SPAA '06: Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, pages 103–112, Cambridge, Massachusetts, USA, 2006. ACM Press.

[38] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of divisible requests. Research report rr-6002, INRIA, 2006. http://hal.inria.fr/inria-00108524. Also available as LIP research report RR2006-19.

[39] M. Litzkow, M. Livny, and M. W. Mutka. Condor - A hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111. IEEE Computer Society Press, 1988.

[40] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. Gehrke. Online scheduling to minimize average stretch. In *IEEE Symposium on Foundations of Computer Science*, pages 433–442, 1999.

[41] J.-F. Pineau, Y. Robert, F. Vivien, and J. Dongarra. Matrix product on heterogeneous master-worker platforms. *13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, February 20-23 2008.

[42] Prime. URL: http://www.mersenne.org.

[43] K. Pruhs, J. rí Sgall, and E. Torng. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter Online Scheduling. Chapman & Hall/CRC Press, 2004.

[44] T. Saif and M. Parashar. Understanding the behavior and performance of non-blocking communications in MPI. In *Proceedings of Euro-Par 2004: Parallel Processing*, LNCS 3149, pages 173–182. Springer, 2004.

[45] A. S. Schulz and M. Skutella. The power of α-points in preemptive single machine scheduling. *Journal of Scheduling*, 5(2):121–133, 2002. DOI:10.1002/jos.093.

[46] SETI. URL: http://setiathome.ssl.berkeley.edu.

[47] B. A. Shirazi, A. R. Hurson, and K. M. Kavi. *Scheduling and load balancing in parallel and distributed systems*. IEEE Computer Science Press, 1995.

[48] O. Sinnen and L. Sousa. Communication contention in task scheduling. *IEEE Trans. Parallel Distributed Systems*, 16(6):503–515, 2004.

[49] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3:59–66, 1956.

[50] T. Yang and A. Gerasoulis. DSC: Scheduling parallel tasks on an unbounded number of processors. *IEEE Trans. Parallel and Distributed Systems*, 5(9):951–967, 1994.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399