Scheduling Concurrent Bag-of-Tasks Applications on Heterogeneous Platforms

Anne Benoit, *Member*, *IEEE*, Loris Marchal, Jean-François Pineau, *Student Member*, *IEEE*, Yves Robert, *Fellow*, *IEEE*, and Frédéric Vivien, *Member*, *IEEE*

Abstract—Scheduling problems are already difficult on traditional parallel machines, and they become extremely challenging on heterogeneous clusters. 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 a heuristic 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.

Index Terms—Scheduling and task partitioning, online computation, parallelism and concurrency, measurement, evaluation, modeling, simulation of multiple-processor systems.

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 application [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 [2] and BOINC-like computations [3]. 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 [4], [5].

In this paper, we still target fully parallel applications, but introduce a much more complex (and more realistic) framework than scheduling a single application. We envision a situation where users, or clients, submit several bag-of-tasks applications 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: 1) from the clients' point of view, the response time (the delay between the arrival of an application and the completion of its last task) is expected to be much smaller and 2) 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 or maximum slowdown. 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

- *A. Benoit and Y. Robert are with ENS Lyon, University of Lyon, and LIP, 46 allee d'Italie, 69007 Lyon, France. E-mail: {Anne.Benoit, Yves.Robert}@ens-lyon.fr.*
- *L. Marchal is with CNRS, University of Lyon, and LIP, 46 allee d'Italie, 69007 Lyon, France. E-mail: Loris.Marchal@ens-lyon.fr.*
- *J.-F. Pineau is with LIRMM, 161 rue Ada 34392 Montpellier Cedex 5 France. E-mail: Jean-Francois.Pineau@lirmm.fr.*
- *F. Vivien is with INRIA, University of Lyon, and LIP, 46 allee d'Italie, 69007 Lyon, France. E-mail: Frederic.Vivien@inria.fr.*

Manuscript received 28 May 2008; revised 18 Jan. 2009; accepted 28 May 2009; published online 24 July 2009.
Recommended for acceptance by F. Lombardi.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-05-0235.
Digital Object Identifier no. 10.1109/TC.2009.117.

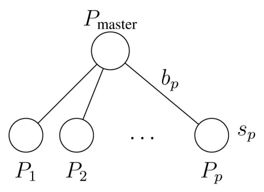

Fig. 1. A star network.

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 bag-of-tasks applications which are submitted online. Our scheduling algorithm relies on complicated mathematical tools but can be computed in time polynomial of 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 a heuristic for the general case of online applications. On the practical side, we have conducted MPI experiments and extensive simulations, 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 Section 4 to the presentation of our heuristic for the online case. In Section 5, we report our set of simulations and MPI experiments, and compare our solution against several classical heuristics from the literature. Section 6 is devoted to an overview of related work. Finally, we state some concluding remarks in Section 7.

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 introduce the maximum stretch objective function.

2.1 Platform Model

We target a heterogeneous master-worker platform (see Fig. 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 (respectively, receive) a message of size X to (respectively, 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.

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 [6], [7], [8]. In particular, a given processor can send an unlimited number of messages in parallel, and each of these messages is routed as if it 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 [9], 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 bidirectional, or full duplex).

Another more restricted model is the *one-port* model [10], [11]. 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 (this hypothesis is not relevant in our study, as the master processor is the only one sending 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 [12]. However, recent multithreaded communication libraries such as MPICH [13], [14] 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 are 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.

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 bag-of-tasks applications 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 [7]. 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 application, 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 (cleanup) periods. It is shown in [4], [15] 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 a solution which maximizes the total throughput: $\rho^{*(k)} = \sum_{u=1}^{p} \rho_u^{*(k)}$.

We write the following linear program (see (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 states 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.

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

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. The first two sets of constraints can be transformed into

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

Then, the last constraint can be rewritten:

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

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 [4], [15] that any feasible schedule under one of the multiport models 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. For example, the following procedure builds an optimal schedule for the BMP-FC-SS model (bounded multiport communication with fluid computation and synchronous start):

- 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 optimal, not only among periodic schedules, but also more generally among all possible schedules. When considering the most constrained BMP-AC model (bounded multiport communication with atomic computation), we have to change the computation policy into:

- Processor P_u processes its tasks one at a time and in the order it has (completely) received each of them.

The execution time of this schedule differs from the minimum execution time by a constant factor, independent of the total number of tasks $\Pi^{(k)}$ to process [4]. 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

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

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 [16] of an application is its 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 earliest time at which all $\Pi^{(k)}$ tasks of A_k are completed. Because there might be other applications running concurrently to A_k during part or whole of its execution, we expect that $MS^{(k)} \ge MS^{*(k)}$. 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

$$\mathcal{S}_k = \frac{MS^{(k)}}{MS_{\text{opt}}^{(k)}} \le \frac{MS^{(k)}}{MS^{*(k)}} = \frac{\rho^{*(k)}}{\rho^{(k)}}.$$

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

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

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 The Offline Case

In this section, we present an asymptotically optimal algorithm for the minimization of the maximum stretch of several bag-of-tasks applications in the offline case, that is, when application release dates and characteristics are known in advance. In this section, we therefore assume that all characteristics of the n applications A_k, $1 \le k \le n$, are known in advance.

3.1 Set of Possible Schedules

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 can then be found using a binary search on possible values.

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

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

Thus, given a candidate value $\mathcal{S}$, we define deadline:

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

for each application A_k, $1 \leq k \leq n$. This means that if each application is completed before its deadline, then the expected max-stretch is reached. 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 $t_j \in \{r^{(1)}, \dots, r^{(n)}\} \cup \{d^{(1)}, \dots, d^{(n)}\}$, such that $t_j \leq t_{j+1}$, $1 \leq j \leq 2n-1$. (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.) 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 $t_{j+1} \leq d^{(k)}$.

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 during that interval.
- $\rho^{(k)}_u(t_j, t_{j+1})$ 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 during that interval.
- $B^{(k)}_u(t_j)$ 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: $\forall\, 1 \leq k \leq n$,

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

- **Nonnegative buffers.** Each buffer should always have a nonnegative size:

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

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

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

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

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

- **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:

$$\begin{aligned} &\forall\, 1 \leq k \leq n, \forall\, 1 \leq j \leq 2n-1, \forall\, 1 \leq u \leq p, \\ &\qquad B^{(k)}_u(t_{j+1}) = B^{(k)}_u(t_j) \\ &\qquad + \big(\rho^{(k)}_{M \to u}(t_j, t_{j+1}) - \rho^{(k)}_u(t_j, t_{j+1})\big) \times (t_{j+1} - t_j). \end{aligned} \tag{7}$$

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

$$\forall\, 1 \leq j \leq 2n-1, \forall\, 1 \leq u \leq p, \sum_{k=1}^{n} \rho^{(k)}_u(t_j, t_{j+1}) \frac{w^{(k)}}{s^{(k)}_u} \leq 1. \tag{8}$$

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

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

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

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

- **Nonnegative throughputs.**

$$\begin{aligned} &\forall\, 1 \leq u \leq p, \forall\, 1 \leq k \leq n, \forall\, 1 \leq j \leq 2n-1, \\ &\qquad \rho^{(k)}_{M \to u}(t_j, t_{j+1}) \geq 0 \text{ and } \rho^{(k)}_u(t_j, t_{j+1}) \geq 0. \end{aligned} \tag{11}$$

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:

$$\begin{cases} \rho^{(k)}_{M\to u}(t_j,t_{j+1}), \rho^{(k)}_{u}(t_j,t_{j+1}), \forall\, k,u,j \text{ such that} \\ 1\le k\le n, 1\le u\le p, 1\le j\le 2n-1, \\ \text{and all previous constraints are satisfied.}\end{cases} \qquad \text{(K)}$$

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

$$\forall\, 1\le j\le 2n-1,\quad B^{(k)}_u(t_{j+1}) - B^{(k)}_u(t_j) = \left(\rho^{(k)}_{M\to u}(t_j,t_{j+1}) - \rho^{(k)}_u(t_j,t_{j+1})\right)\times(t_{j+1}-t_j).$$

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

$$B^{(k)}_u(t_{\text{stop}}) - B^{(k)}_u(t_{\text{start}}) = \sum_{\substack{[t_j,t_{j+1}]\\ t_j\ge r^{(k)}\\ t_{j+1}\le d^{(k)}}} \left(\rho^{(k)}_{M\to u}(t_j,t_{j+1}) - \rho^{(k)}_u(t_j,t_{j+1})\right)\times(t_{j+1}-t_j).$$

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

$$\sum_{\substack{[t_j,t_{j+1}]\\ t_j\ge r^{(k)}\\ t_{j+1}\le d^{(k)}}} \rho^{(k)}_{M\to u}(t_j,t_{j+1})\times\left(t_{j+1}-t_j\right) = \sum_{\substack{[t_j,t_{j+1}]\\ t_j\ge r^{(k)}\\ t_{j+1}\le d^{(k)}}} \rho^{(k)}_u(t_j,t_{j+1})\times\left(t_{j+1}-t_j\right).$$

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

$$\sum_{u=1}^{p}\sum_{\substack{[t_j,t_{j+1}]\\ t_j\ge r^{(k)}\\ t_{j+1}\le d^{(k)}}} \rho^{(k)}_u(t_j,t_{j+1})\times\left(t_{j+1}-t_j\right) = \Pi^{(k)}.$$

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

3.3 Bounding Buffer Sizes

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

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

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 proved that limiting the buffer size for independent tasks scheduling leads to NP-complete problems [17].

3.4 Equivalence between Nonemptiness of Polyhedron K and Achievable Stretch

Finding a point in Polyhedron K allows us to determine whether the candidate value for the stretch is feasible. Depending on whether Polyhedron K is empty, the binary search is 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 is described below. For now, we concentrate on stating 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 are slightly more involved, as we will see in Section 3.6. However, this technique allows us to approach optimality.

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

The detailed proof of this result is available in the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117. It consists first in proving that any schedule must satisfy the constraints defining Polyhedron K, and then, in building a valid schedule from a given point in the polyhedron.

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.

3.5 Binary Search

To find the optimal stretch, we perform a binary search. We first present a simple approximated search using the emptiness of Polyhedron (K) to determine whether it is possible to achieve the current stretch. Then, we present an optimal but more involved search.

The lower bound on the achievable stretch is 1. The initial upper bound for this binary search is also quite naive. 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. 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

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

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.

Algorithm 1: Binary search

```
begin
    $\mathcal{S}_{\text{inf}} \leftarrow 1$
    $\mathcal{S}_{\text{sup}} \leftarrow \mathcal{S}_{\max}$
    while $\mathcal{S}_{\text{sup}} - \mathcal{S}_{\text{inf}} > \epsilon$ do
        $\mathcal{S} \leftarrow (\mathcal{S}_{\text{sup}} + \mathcal{S}_{\text{inf}})/2$
        if Polyhedron (K) is empty then
            $\mathcal{S}_{\text{inf}} \leftarrow \mathcal{S}$
        else
            $\mathcal{S}_{\text{sup}} \leftarrow \mathcal{S}$
    return $\mathcal{S}_{\text{sup}}$
end
```

Suppose that we are given $\epsilon > 0$. The binary search is conducted using Algorithm 1. This algorithm approaches the optimal stretch, as stated by the following theorem. The proof of the theorem is available in the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117.

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

In fact, one can find the optimal stretch in polynomial time. First, recall that the application deadlines are defined by the application release dates and the targeted stretch $\mathcal{S}$:

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

Each deadline is thus an affine function in $\mathcal{S}$, as depicted in Fig. 2. We call *critical values* of the stretch the values for which the relative ordering of the application release dates and deadlines changes,

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

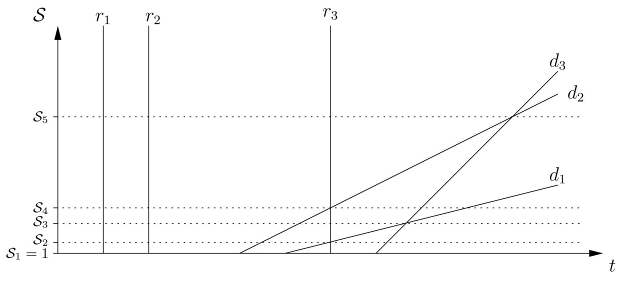

Fig. 2. Relation between stretch and deadlines.

To simplify our notations, we add two artificial *critical values* corresponding to our bounds on the stretch: $\mathcal{S}_1 = 1$ and $\mathcal{S}_m = \mathcal{S}_{\max}$.

Our goal is to find the optimal stretch by slicing the stretch space into the intervals defined by the *critical values*. Within each interval, the deadlines are linear functions of the stretch (and release dates are constant). 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.

3.5.1 Within a Stretch Interval

In the following, we work on one stretch interval, called as $[\mathcal{S}_a, \mathcal{S}_b]$. 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 $\{t_j\}_{j=1\dots2n} = \{r^{(k)}, d^{(k)}\}$, with $t_j \le t_{j+1}$. As the values of the t_js may change when $\mathcal{S}$ varies, we write $t_j = \alpha_j\mathcal{S} + \beta_j$. This notation is general enough:

- 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, two t_js may be equal, especially for critical values ($\mathcal{S} = \mathcal{S}_a$ or $\mathcal{S} = \mathcal{S}_b$). 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 the above 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 $A_{M\to u}^{(k)}(t_j, t_{j+1})$ 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 $A_u^{(k)}(t_j, t_{j+1})$ 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

$$A_{M\to u}^{(k)}(t_j, t_{j+1}) = \rho_{M\to u}^{(k)}(t_j, t_{j+1}) \times (t_{j+1} - t_j),$$
$$A_u^{(k)}(t_j, t_{j+1}) = \rho_u^{(k)}(t_j, t_{j+1}) \times (t_{j+1} - t_j),$$

with $t_{j+1} - t_j = (\alpha_{j+1} - \alpha_j)\mathcal{S} + (\beta_{j+1} - \beta_j)$. All constraints can be rewritten with these new notations; the two instances below exemplify all rewriting cases:

- **Task conservation.**

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

- **Bounded computing capacity.**

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

We finally add a constraint to force the objective stretch to be in the targeted stretch interval:

$$\mathcal{S}_a \le \mathcal{S} \le \mathcal{S}_b. \qquad (12)$$

We thus rewrite as above all the constraints defining Polyhedron (K), and then, we add the new constraint 12. This way, we obtain a linear program enabling us to check what is the minimal achievable stretch in the interval $[\mathcal{S}_a, \mathcal{S}_b]$, if any.

3.5.2 Overall Binary Search

The linear program we just described is used as a building brick for our exact binary search. For the 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 no better possible stretch in $[\mathcal{S}_a, \mathcal{S}_b]$, and thus, there is no better stretch overall. 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 may be 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 than or equal to $\mathcal{S}_b$, and the binary search is continued with larger stretch values. As the number of critical values is at worst quadratic in the number of applications, the overall binary search runs in time polynomial in the size of the problem. (For more details, see the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117.)

3.6 Quasi Optimality for More Realistic Bounded Multiport Models

In this section, we briefly explain how the previous optimality result can be adapted to the other bounded multiport models presented in Section 2.1.3 (refer to the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117, for a detailed description of all technical results and their proofs). We detail the case of the one-port model in the next section. 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 S_1 which is optimal under the fluid model (BMP-FC-SS), the idea is to build a schedule S_2 with comparable performance under a more constrained scenario.

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.

From a schedule valid under the totally fluid model (BMP-FC-SS), we can build a schedule

- under the BMP-FC model, where the maximum lateness for each application is

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

- under the BMP-AC model where the maximum lateness for each application is

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

We are then able to 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 negligible 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 task characteristics, then the problem looks totally different: any schedule is running for a very long time, and the time separating the release dates is 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 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 3. *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 (respectively, BMP-AC) schedule, and $\mathcal{S}_{\text{opt}}$ the stretch of the optimal fluid schedule.

The proof of this theorem is available in the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117 (as *Theorem 7*).

3.7 Asymptotic Optimality for the One-Port Model

To establish an asymptotic optimality result for the one-port model, we: 1) modify constraints to deal with the one-port communication model instead of the bounded multiport one; 2) transform a fluid schedule into an atomic one where file transfers (and task computations) are serialized and applications do not terminate much later than in the reference fluid schedule; 3) prove that the obtained schedule, valid under the one-port model, is asymptotically optimal: when the granularity of tasks tends to zero, the achieved maximum stretch tends to the optimal one.

3.7.1 Modifying Constraints to Deal with the One-Port Communication Model

We cannot simply extend the results obtained for the fluid model to the one-port model 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 then replaced by

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

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):

$$\begin{cases} \rho_{M \to u}^{(k)}(t_j, t_{j+1}), \rho_u^{(k)}(t_j, t_{j+1}), \forall\, k, u, j \text{ such that} \\ 1 \leq k \leq n, 1 \leq u \leq p, 1 \leq j \leq 2n-1, \text{and } (3), (4), \\ (5), (6), (7), (8), (9), (10\text{-b}), \text{and } (11) \text{ are satisfied.} \end{cases} \qquad (\text{K}_1)$$

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. To do that, we need some special schedule transformations.

3.7.2 Basic Schedule Transformations

We define two schedule transformations taking as input a fluid schedule S_{fluid} scheduling n applications $A_1, \dots, A_n$ on a *single* resource. These transformations have the following properties: With the first transformation, no task T terminates later under the modified schedule than under S_{fluid}. With the second transformation, no task T starts earlier under the modified schedule than under S_{fluid}. (All details and proofs are available in the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117.) These two transformations are defined as follows: We denote by t_k the time needed by the resource to process one task of A_k at full speed. Under fluid schedule S_{fluid}, each application A_k is devoted a share α_k of the resource such that $\sum_{k=1}^{n} \alpha_k \leq 1$. From S_{fluid}, we build an atomic model schedule S_{1D} using a 1D load-balancing algorithm [18]: at any time step, if n_k is the number of tasks of application A_k already scheduled, the next task to be scheduled is the one minimizing $\frac{(n_k+1) \times t_k}{\alpha_k}$.

We can prove that under schedule S_{1D}, a task T does not terminate later than under S_{fluid}. S_{1D} 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 starts 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. From S_{fluid} of makespan M, we build a schedule S_{fluid}^{-1} by reversing time: a task beginning at time b and finishing at time f in S_{fluid} is scheduled to start at time $M - f$ and to terminate at time $M - b$ in S_{fluid}^{-1}, and is processed at the same rate as in S_{fluid}.
2. We apply the 1D load-balancing algorithm [18] to S_{fluid}^{-1}, leading to schedule S_{1D}^{-1}. We know that a task T does not terminate later in S_{1D}^{-1} than in S_{fluid}^{-1}.
3. We transform S_{1D}^{-1} by reverting time one last time: we obtain the schedule S_{1D}^{-2}. A task beginning at time b and finishing at time f in S_{1D}^{-1} starts at time $M - f$ and finishes at time $M - b$ in S_{1D}^{-2}. We can prove that under schedule S_{1D}^{-2}, a task T does not terminate sooner than under S_{fluid}. Note that S_{1D}^{-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 1D schedule, to avoid to start a task too early.

3.7.3 Schedule for the One-Port Model

Theorem 4. *1) If there exists a schedule valid under the one-port model with stretch $\mathcal{S}_1$, then Polyhedron (K_1) is not empty for $\mathcal{S}_1$ and 2) 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 3.6, whose stretch $\mathcal{S}$ is such that $\lim_{g \to 0} \mathcal{S} = \mathcal{S}_2$.*

The nontrivial part of the theorem is the second one. We now sketch how to construct such a schedule. (All details can be found in the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117, as *Theorem 8*.)

We start from a point in Polyhedron (K_1). During each interval $[t_j, t_{j+1}]$ and for each worker P_u, we proceed as follows:

1. We define a fluid model schedule S_f from the point in Polyhedron (K_1): the tasks of any application are sent and processed at the rates defined by the point in (K_1).
2. We transform both the communication schedule and the computation schedule using 1D load-balancing algorithms. We first round down the number of

tasks of A_k that are sent during interval $[t_j, t_{j+1}]$ to P_u, $n^{\text{comm}}_{u,j,k}$. The number of tasks $n^{\text{comp}}_{u,j,k}$ that can be computed on P_u during $[t_j, t_{j+1}]$ 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.

The first $n^{\text{comm}}_{u,j,k}$ tasks sent in schedule S_f are organized with the 1D load-balancing algorithm into S_{1D}, while the last $n^{\text{comp}}_{u,j,k}$ tasks executed in schedule S_f are organized with the inverse 1D load-balancing algorithm S^{-2}_{1D}. This gives us a schedule for the sending of $n^{\text{comm}}_{u,j,k}$ tasks and the computation of $n^{\text{comp}}_{u,j,k}$ tasks.

3. Next, computations are shifted: for each application A_k, the computation of the first task of A_k is discarded (the processor is kept idle instead of computing this task), and we replace the computation of task i by the computation of task $i-1$.
4. Finally, at time $d^{(k)}$, some tasks of application A_k are still not processed, and some may even not have been received yet. We serialize the sending operations of all the yet missing tasks. Then, some tasks remain to be processed on each processor. We then compute (at full speed) all these tasks.

The proof of validity of the obtained schedule comes from the fact that a task does not start earlier in S^{-2}_{1D} than in S_f, and completes no later in S_{1D} than in S_f. Therefore, the data needed for the execution of a given task are received in time. To assess its performance, we show that the lateness of any application A_k is at most:

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

Therefore, as in Theorem 3, when the granularity becomes small, the stretch of the obtained schedule becomes as close as we want to that of the schedule we started from.

4 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 reassigned 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 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 $\Pi^{(k)}_{\text{rem}}$ 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 thus, have $\Pi^{(k)}_{\text{rem}} \neq 0$ for all applications. For the new application, we have $\Pi^{(k_{\text{new}})}_{\text{rem}} = \Pi^{(k_{\text{new}})}$. We also consider as parameters the state $B^{(k)}_u(T_{\text{new}})$ of the buffers at time T_{new}. We also have $B^{(k_{\text{new}})}_u(T_{\text{new}}) = 0$.

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 nonempty 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 ordered deadlines: $\{t_j\} = \{d^{(k)}\} \cup \{T_{\text{new}}\}$ such that $t_j \le t_{j+1}$. The constraints are the same as the ones used for Polyhedron K, except the constraint on the number of tasks 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 us 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.

5 MPI Experiments and SimGrid Simulations

We have conducted several experiments in order to compare our algorithms to reference scheduling strategies. We first present the reference scheduling heuristics, and then, 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/.

5.1 Reference Scheduling Heuristics

In this section, we present strategies that are able to schedule multiple applications in an online setting. Most of these strategies are simple and wait for an application to terminate before scheduling another application. Although these strategies may be far from the optimal scheduling in a number of cases, they are representative of existing Grid schedulers. We first outline policies for selecting the set of applications to be executed:

- First in first out **(FIFO).** Applications are computed in the order of their release dates.
- Shortest processing time **(SPT).** When an application terminates (or the first application is released), the application with the smallest processing time is scheduled (the processing time is approximated by MS^*, see Section 2.3).
- Shortest remaining processing time **(SRPT).** At each release date or termination date, the application with the smallest remaining processing time is scheduled. The remaining processing time is the time needed to process the remaining tasks of the application (and is approximated as previously).

- Shortest weighted remaining processing time **(SWRPT).** This strategy is very similar to $SRPT$, but the remaining processing time of the released applications is weighted with MS^*, that is the application with the smallest ratio between the remaining processing time and the total processing time is scheduled first. In practice, it gives small applications, i.e., in term of MS^*, a priority against large applications which are almost finished, which is better in order to minimize the ratio between the response time of the application and the time this application would have spent if executed alone (MS^*).

The importance and relevance of the above heuristics are outlined in Section 6. Once an application is selected, several policies exist for scheduling its tasks onto the platform:

- Round-robin **(RR).** All workers are selected in a cyclic way.
- Minimum completion time **(MCT).** Given a task of the application, we select the worker that will finish this task the earliest, given the current load of the platform.
- Demand driven **(DD).** 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 12 different greedy algorithms. We also test a more sophisticated algorithm:

- Master worker for multiple applications **(MWMAs).** This algorithm computes on each time interval a steady-state strategy to schedule the available applications, as presented in [19]. 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: 1) the remaining number of tasks of the applications (variant called $MWMA\_NBT$) or 2) the remaining processing time of the applications (variant called $MWMA\_MS$). Both variants are compared in the experiments.

In addition to the previous scheduling strategies, we have implemented several heuristics based on our static algorithm, called Clever Burst Steady-State Stretch Minimization (**CBS3M**) in the following. As emphasized in Section 3.7.3, the fluid solution of the $CBS3M$ algorithm needs to be adapted to cope with the one-port model. Rather than literally implementing Section 3.7.3, which is best suited to compute theoretical bounds, we first implement a 1D load-balancing algorithm for the master's sending operations, and then, we test two variants for the workers to choose the next task to compute among those they have received: FIFO and Earliest Deadline First (**EDF**). In such a way, we first implemented the online version of the algorithm, as described in Section 4, which gives the strategy CBS3M_$*$_ONLINE (with $*$=FIFO|EDF). As a comparison basis, we also add a strategy, CBS3M_$*$_ROFF ($*$=FIFO|EDF) with all information about future submissions: this strategy runs the $CBS3M$ algorithm under a rounded offline model: the algorithm has a complete information and computes the whole schedule at the beginning (as described in Section 3), but is then adapted (rounded) to the one-port model.

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 [20].

5.2 Experimental Settings

In order to test and compare our heuristics, we perform both simulations and real experiments. Simulations are conducted using the SimGrid [21] simulator, while experiments make use of the MPICH-2 communication library [22].

Communication and computation sizes are generated by transmitting random data, or by computing random matrix products, as described below; no real application is used. In particular, this allows us to emulate a heterogeneous platform: we have full freedom to slow down some communications by transmitting the same data several times, and some computations by performing several times the same task.

The use of MPI to perform communications leads us to serialize the communications, and to abandon the multiport model in favor of the one-port model. Thus, we use the adaptation of our theoretical study for the one-port model (as described in the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117) in order to determine the optimal max-stretch and the solutions of the CBS3M strategies.

Our theoretical study is fully general, allowing computing times to be unrelated: a processor can process different applications with different speeds. However, for sake of simplicity, we consider in the experiments and the simulations that we have uniform processors: the processing time of a task depends only on its size (depending on the application) and the speed of the processor, not on the application. As we target a heterogeneous master-worker platform, we generate several platform scenarios. The computing speeds are uniformly distributed in interval $[\alpha, 10.\alpha]$, where α is the reference speed. Similarly, the link bandwidths are uniformly distributed in interval $[\beta, 10.\beta]$, where β is the reference bandwidth.

The experiments are 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.

Even if we totally control the platform parameters (computing speeds and bandwidths), when these characteristics are needed by a heuristic to take scheduling decisions, the parameters are measured within the program by sending a small message, or performing a small task. This is true both in the MPI implementation and the simulations.

The time needed to measure the platform characteristics and take scheduling decisions is taken into account in the experiments (but not in the simulations). This phase usually takes a few seconds in the experiments (up to 1 minute) for scenarios of a few hours, and thus, represents less than 1 percent of the total running time.

5.3 Applications

A bag-of-tasks application is described by its release date, its number of tasks, and the communication and computation sizes of each task. For our experiments and simulations,

TABLE 1
Parameters for the MPI Experiments and for the SimGrid Simulations

	parameter	experiments	simulations
general	number of workers	8	10
	number of applications	12	20
arrival dates	mean of the distribution in the log space	4.0	4.0
	standard deviation in the log space	1.2	1.2
computations	maximum amount of work application (Gflops)	76.8	409
	minimum amount of work per task (Gflops)	3.1	3.1
communications	maximum amount of communication per application (MB)	800	6,000
	minimum amount of communication per task (MB)	40	40
number of tasks	minimum number of tasks per application	10	20

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 [23].
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 Table 1.

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

5.4 Results

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

5.4.1 Simulation Results

In this section, we detail the results of the simulations. We run 1,000 simulations based on the parameters described in Table 1. Fig. 3 presents the results of all heuristics for the max-stretch metric, whereas Fig. 4 shows the evolution of some heuristics (the best ones) over the load of the scenario. Here, the load is characterized with the optimal theoretical achievable max-stretch in the fluid model: 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. All results are relative to the optimal max-stretch, which is computed in the offline case. A relative max-stretch of 1.5 means that the corresponding strategies achieve a max-stretch which is 1.5 times the optimal one, thus with a degradation of 50 percent.

The CBS3M heuristics perform very well for the max-stretch: CBS3M_EDF_ONLINE achieves the best max-stretch between all heuristics in 64 percent of the simulations. This heuristic performs significantly better than all other heuristics: it has an average max-stretch of 1.163 times the optimal max-stretch, the lowest standard deviation (0.118), and the minimum worst case (1.93) among all heuristics.

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

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_ROFF	1.022	1.379	(± 0.276)	3.74	(the best in 3.8 %)
CBS3M_EDF_ROFF	**1.011**	**1.213**	**(± 0.125)**	**2.06**	**(the best in 26.2 %)**

Fig. 3. Simulation results: Relative max-stretch of all heuristics in the simulations.

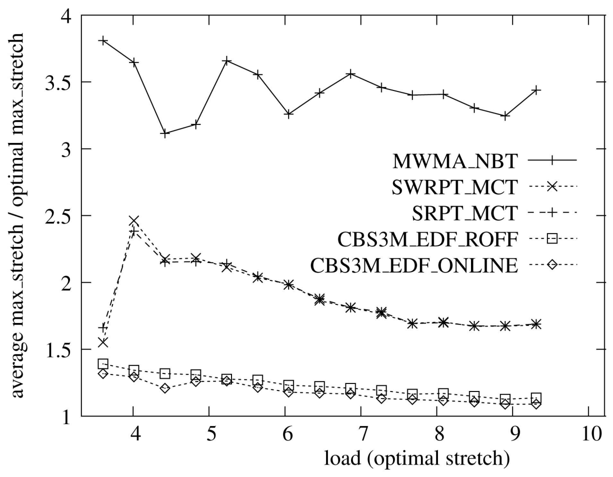

Fig. 4. Simulation results: Evolution of the relative max-stretch of best heuristics in the simulations under different load conditions.

(except MWMA) are limited to scheduling only one application at a time, leading to an overall bad utilization of the computing platform.

In Fig. 4, one can notice that surprisingly, the offline version of CBS3M 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.

We also observe that resource selection is important on heterogeneous platforms, as the strategies which have the worst relative max-stretch are the ones using round-robin or demand-driven policies.

Another observation is the relatively bad results of the involved MWMA strategies (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.

In the Web supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TC.2009.117, we plot the results of the best heuristics for other objectives: sum-stretch, makespan, max, and sum-flow. Quite 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.

5.4.2 Experimental Results

We now move to the real experiments with MPI communications. The experiments were performed on 50 different platform and application settings. As several heuristics performed very poorly in the simulations, especially the heuristics based on round-robin and demand-driven policies, and thus, would have lead to huge computation times, we discarded them and restricted ourselves to a smaller set of heuristics in order to get reasonable running times.

Once again, the performance of a given strategy is measured through its relative max-stretch that is the ratio between the obtained max-stretch and the theoretical optimal max-stretch in the fluid model.

The results of the experiments are summarized in Fig. 5; Fig. 6 presents the results for the best four strategies: CBS3M using EDF policy, in both the offline and online versions, MWMA_NBT and SWRPT. They are quite similar to the simulation results: the four versions of CBS3M achieve a better relative max-stretch than most other strategies. Once again the online version performs generally better than the offline version, as explained earlier. The major difference concerns the MWMA strategies, which perform much better than in the simulations. This can be explained by the different scenarios used in experiments and simulations: in order to avoid huge running times in the experiments, we concentrate on simple scenarios, with smaller applications, whereas in the simulations, we use larger applications as simulations run for a short time even with long simulated running times. To fully assess the adequacy of the simulations and the experiments, we decided to rerun the experimental scenarios within our simulator, and to compare both results.

5.4.3 Simulations on Experimental Platforms

In this section, we check the accuracy of our simulations, by "simulating the experiments": we run simulations on the

Algorithm	minimum	average	(± stddev)	maximum	(fraction of best result)
CBS3M_EDF_ROFF	**1.04**	**1.30**	**(± 0.13)**	**1.63**	**(the best in 38.0%)**
CBS3M_EDF_ONLINE	**1.02**	**1.41**	**(± 0.30)**	**2.09**	**(the best in 30.0%)**
CBS3M_FIFO_ROFF	1.04	1.38	(± 0.28)	2.97	(the best in 12.0%)
CBS3M_FIFO_ONLINE	1.02	1.46	(± 0.26)	1.96	(the best in 6.0%)
FIFO_MCT	1.10	1.81	(± 0.60)	4.15	(the best in 4.0%)
FIFO_RR	1.35	4.99	(± 3.46)	19.50	(the best in 0.0%)
MWMA_MS	1.22	2.29	(± 0.56)	4.05	(the best in 0.0%)
MWMA_NBT	1.13	1.50	(± 0.17)	2.06	(the best in 4.0%)
SPT_DD	1.33	4.87	(± 3.10)	18.75	(the best in 0.0%)
SPT_MCT	1.08	1.84	(± 0.61)	3.43	(the best in 4.0%)
SRPT_MCT	1.09	1.87	(± 0.59)	3.38	(the best in 0.0%)
SWRPT_MCT	1.08	1.88	(± 0.59)	3.38	(the best in 2.0%)

Fig. 5. MPI experiment results: Relative max-stretch of selected heuristics in the experiments.

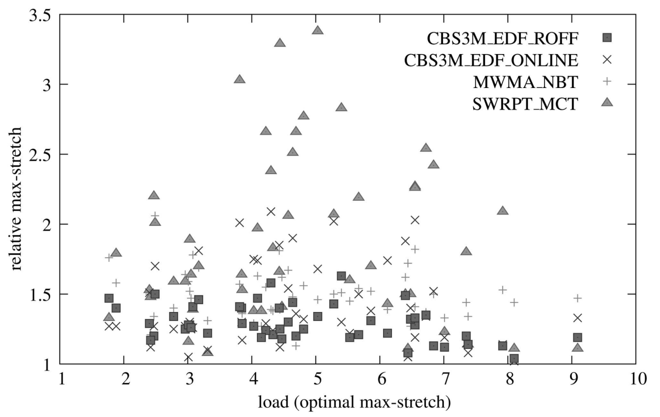

Fig. 6. MPI experiment results: Evolution of the relative max-stretch of the best heuristics in the simulations under different load conditions.

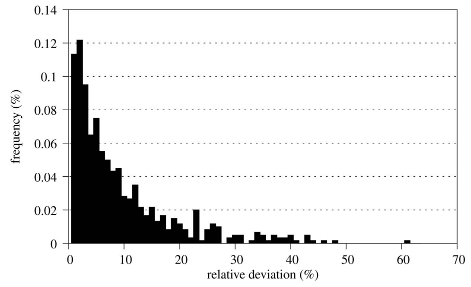

Fig. 7. Distribution of relative deviation between simulations and experiments.

same scenarios (platforms and application parameters) that have been used for real experiments. Obviously, the executions will differ, and we do not expect the results to be strictly identical: the simulations do not account for the dynamic nature of the platform used in real experiments. Simulations do not take scheduling times into account and rely on exact application/platform parameters, while experiments can only rely on inaccurate predicted values.

In Fig. 7, we plot the distribution of the relative deviation between the max-stretch obtained in the experiments and the max-stretch obtained in the simulations, for all strategies. The maximum deviation is 60.1 percent, but the average deviation is only 8.9 percent, with a standard deviation of 9.5 percent (the median value is 5.5 percent). Overall, the accuracy of the simulations is satisfactory, and even good if we keep in mind all possible sources of differences between simulations and experiments.

6 Related Work

Related literature can be classified into three main categories: 1) bag-of-tasks applications; 2) steady-state scheduling; and 3) flow-type objective functions and online scheduling.

1. *Bag-of-tasks applications:* Bag-of-tasks applications are parallel applications whose tasks are all independent. Their study is motivated by problems that are addressed by collaborative computing efforts. Their use goes from the pioneering project SETI@home [24], to recent and active projects like OurGrid [25] or BOINC [3]. Bag-of-tasks applications are well suited for computational grids, because communication can easily become a bottleneck for tightly coupled parallel applications. The use of bag-of-tasks applications includes user-centric approaches like APST [26] and system-centric approaches able to run multiple applications, like Condor [27]. Most work on scheduling bag-of-task applications considers a single application [28], [29], [30]. As in our study, Anglano and Canonico [31] consider several applications arriving over time and target a flow-based objective (sum-flow). However, communications are not taken into account in [31] and the approach is knowledge-free when we assume that we have a good knowledge on applications and try to make the most of it.
2. *Steady-state scheduling:* While minimizing the makespan is an NP-hard problem in most practical situations [32], 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 [33], and has been used successfully in many situations [34]. In particular, steady-state scheduling has been used to schedule independent tasks on heterogeneous tree-overlay networks [4], and adapted to cope with multiple applications [19].
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 monoprocessor 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.

6.1 Flow Optimization

On a single processor, the max-flow is optimized by *First-Come First-Serve* (FCFS) (see Bender et al. [16], for example), and the sum-flow is optimized by the *shortest remaining processing time first* (SRPT) [35].

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 [36]. In other words, the two objective functions cannot be optimized simultaneously to obtain a nontrivial competitive factor (FCFS is not taking into account the weight of tasks in the objective).

6.2 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 et al. [37] designed a Polynomial Time Approximation Scheme (PTAS) for minimizing the sum-stretch with preemption. Chekuri and Khanna [38] 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 [36].

6.3 Max-Stretch Minimization

Max-stretch can be optimally minimized in the offline case [36], 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 [39]. However, as soon as there are at least three task sizes, no algorithm has 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 [36].

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 subtasks when in the context of this 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 [36], while they play a major role in our case.

7 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 offline 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 multiapplications strategies. Experiments were both run on a real cluster, using MPI, and conducted through extensive simulations, using SimGrid. Both experimental comparisons show a great improvement when using our CBS3M strategy, which achieves an average worse max-stretch only 16 percent greater than the offline optimal max-stretch. To the best of our knowledge, this work is the first attempt to provide efficient scheduling techniques for multiple bag-of-tasks applications in an online scenario.

Future work includes extending the approach to other communication models (such as the contention model of [40]) and more general platforms (such as multilevel trees). It would also be very interesting to deal with more complex applications, whose dependence graphs could be simple pipeline or fork graphs, or even general DAGs. Another direction is to investigate more dynamic settings, where each computing resource could be enrolled in several volunteer grids that compete (or cooperate?) for "their" applications, and where both application and platform parameters are subject to some uncertainties. The lessons learned in this study (such as the usefulness of sharing several applications on the same resource) should prove valuable to tackle these important but difficult problems.

References

[1] M. Adler, Y. Gong, and A.L. Rosenberg, "Optimal Sharing of Bags of Tasks in Heterogeneous Clusters," *Proc. 15th ACM Symp. Parallelism in Algorithms and Architectures (SPAA '03),* pp. 1-10, 2003.

[2] H. Casanova and F. Berman, "Parameter Sweeps on the Grid with APST," *Proc. Grid Computing: Making the Global Infrastructure a Reality,* F. Berman, G. Fox, and T. Hey, eds., 2003.

[3] "BOINC: Berkeley Open Infrastructure for Network Computing," http://boinc.berkeley.edu, 2009.

[4] 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 and Distributed Systems,* vol. 15, no. 4, pp. 319-330, Apr. 2004.

[5] J. Dongarra, J.-F. Pineau, Y. Robert, and F. Vivien, "Matrix Product on Heterogeneous Master-Worker Platforms," *Proc. ACM SIGPLAN,* pp. 53-62, 2008.

[6] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Trans. Parallel and Distributed Systems,* vol. 5, no. 9, pp. 951-967, Sept. 1994.

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

[8] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 260-274, Mar. 2002.

[9] B. Hong and V. Prasanna, "Distributed Adaptive Task Allocation in Heterogeneous Computing Environments to Maximize Throughput," *Proc. Int'l Symp. Parallel and Distributed Processing (IPDPS '04),* 2004.

[10] P. Bhat, C. Raghavendra, and V. Prasanna, "Efficient Collective Communication in Distributed Heterogeneous Systems," *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS '99),* pp. 15-24, 1999.

[11] P. Bhat, C. Raghavendra, and V. Prasanna, "Efficient Collective Communication in Distributed Heterogeneous Systems," *J. Parallel and Distributed Computing,* vol. 63, no. 3, pp. 251-263, 2003.

[12] T. Saif and M. Parashar, "Understanding the Behavior and Performance of Non-Blocking Communications in MPI," *Proc. Euro-Par 2004: Parallel Processing,* pp. 173-182, 2004.

[13] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing,* vol. 22, no. 6, pp. 789-828, Sept. 1996.

[14] N.T. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," *J. Parallel and Distributed Computing,* vol. 63, no. 5, pp. 551-563, 2003.

[15] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert, "Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms," *Proc. Int'l Symp. Parallel and Distributed Processing (IPDPS '02),* 2002.

[16] M.A. Bender, S. Chakrabarti, and S. Muthukrishnan, "Flow and Stretch Metrics for Scheduling Continuous Job Streams," *Proc. Symp. Discrete Algorithms (SODA '98),* pp. 270-279, 1998.

[17] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Independent and Divisible Tasks Scheduling on Heterogeneous Star-Shaped Platforms with Limited Memory," *Proc. Euromicro Conf. Parallel, Distributed and Network-Based Processing (PDP '05),* pp. 179-186, 2005.

[18] P. Boulet, J. Dongarra, Y. Robert, and F. Vivien, "Static Tiling for Heterogeneous Computing Platforms," *Parallel Computing,* vol. 25, no. 5, pp. 547-568, 1999.

[19] 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 and Distributed Systems,* vol. 19, no. 5, pp. 698-709, May 2008.

[20] "GNU Linear Programming Kit," http://www.gnu.org/software/glpk/, 2009.

[21] A. Legrand, L. Marchal, and H. Casanova, "Scheduling Distributed Applications: The SIMGRID Simulation Framework," *Proc. IEEE/ACM Int'l Symp. Cluster Computing and the Grid (CCGrid '03),* pp. 138-145, May 2003.

[22] W. Gropp, "MPICH2: A New Start for MPI Implementations," *Proc. European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface,* pp. 37-42, 2002.

[23] D.G. Feitelson, *Workload Characterization and Modeling Book.* John Wiley and Sons, http://www.cs.huji.ac.il/feit/wlmod/, 2008.

[24] SETI, http://setiathome.ssl.berkeley.edu, 2009.

[25] 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," *Proc. Int'l Conf. Parallel Processing (ICCP '03),* Oct. 2003.

[26] 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 and Distributed Systems,* vol. 14, no. 4, pp. 369-382, Apr. 2003.

[27] M. Litzkow, M. Livny, and M.W. Mutka, "Condor—A Hunter of Idle Workstations," *Proc. Eighth Int'l Conf. Distributed Computing Systems.* pp. 104-111, 1988.

[28] F.A. da Silva, S. Carvalho, and E.R. Hruschka, "A Scheduling Algorithm for Running Bag-of-Tasks Data Mining Applications on the Grid," *Proc. Euro-Par 2004: Parallel Processing,* pp. 254-262, 2004.

[29] C. Weng and X. Lu, "Heuristic Scheduling for Bag-of-Tasks Applications in Combination with QoS in the Computational Grid," *Future Generation Computer Systems,* vol. 21, no. 1, pp. 271-280, 2005.

[30] A. Sulistio and R. Buyya, "A Time Optimization Algorithm for Scheduling Bag-of-Task Applications in Auction-Based Proportional Share Systems," *Proc. 17th Int'l Symp. Computer Architecture and High Performance Computing (SBAC-PAD '05),* pp. 235-242, 2005.

[31] C. Anglano and M. Canonico, "Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids: A Knowledge-Free Approach," *Proc. Second Int'l Workshop Desktop Grids and Volunteer Computing Systems (PCGRID '08) Workshop Colocated with Int'l Symp. Parallel and Distributed Processing (IPDPS '08),* 2008.

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

[33] D. Bertsimas and D. Gamarnik, "Asymptotically Optimal Algorithms for Job Shop Scheduling and Packet Routing," *J. Algorithms,* vol. 33, no. 2, pp. 296-318, 1999.

[34] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Steady-State Scheduling on Heterogeneous Clusters," *Int'l J. Foundations of Computer Science,* vol. 16, no. 2, pp. 163-194, 2005.

[35] K. Baker, *Introduction to Sequencing and Scheduling.* Wiley, 1974.

[36] A. Legrand, A. Su, and F. Vivien, "Minimizing the Stretch When Scheduling Flows of Divisible Requests," *J. Scheduling,* vol. 11, no. 5, pp. 381-404, 2008.

[37] M.A. Bender, S. Muthukrishnan, and R. Rajaraman, "Approximation Algorithms for Average Stretch Scheduling," *J. Scheduling,* vol. 7, no. 3, pp. 195-222, 2004.

[38] C. Chekuri and S. Khanna, "Approximation Schemes for Preemptive Weighted Flow Time," *Proc. 34th Ann. ACM Symp. Theory of Computing,* pp. 297-305, 2002.

[39] S. Muthukrishnan, R. Rajaraman, A. Shaheen, and J. Gehrke, "Online Scheduling to Minimize Average Stretch," *Proc. IEEE Symp. Foundations of Computer Science,* pp. 433-442, 1999.

[40] O. Sinnen and L. Sousa, "Communication Contention in Task Scheduling," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 6, pp. 503-515, June 2004.

Anne Benoit received the PhD degree from the Polytechnical Institute of Grenoble (INPG) in 2003. From 2003 to 2005, she was a research associate in the School of Informatics, University of Edinburgh, United Kingdom. She is currently an associate professor at the Laboratoire de l'Informatique du Parallélisme, École Normale Supérieure, Lyon, France. Her research interests include performance evaluation, high-level parallel programming, and algorithms and scheduling for distributed heterogeneous platforms. She is the author of more than 15 papers published in international journals, and more than 35 papers published in international conferences. She is a member of the IEEE.

Loris Marchal received the PhD degree from École Normale Supérieure de Lyon in 2006. He is currently a CNRS researcher in the Computer Science Laboratory (Laboratoire de l'Informatique du Parallélisme) at ENS Lyon. His research interest includes parallel algorithm design for heterogeneous platforms and scheduling.

Jean-François Pineau received the PhD degree from École Normale Supérieure de Lyon in 2008. He is currently a postdoctorant in the Computer Science Laboratory LIRM at Montpellier. He is mainly interested in the design of parallel algorithms for heterogeneous platforms and in scheduling techniques. He is a student member of the IEEE.

Yves Robert received the PhD degree from the Institut National Polytechnique de Grenoble in 1986. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of five books, more than 100 papers published in international journals, and more than 150 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for multicore processors, clusters, and grids. He served on many editorial boards, including the *IEEE Transactions on Parallel and Distributed Systems*. He was the program chair of HiPC '06 in Bengaluru and IPDPS '08 in Miami. He is a fellow of the IEEE. He has been elected a senior member of the Institut Universitaire de France in 2007.

Frédéric Vivien received the PhD degree from École Normale Supérieure de Lyon in 1997. From 1998 to 2002, he was an associate professor at Louis Pasteur University, Strasbourg, France. He spent the year 2000 working in the Computer Architecture Group of the MIT Laboratory for Computer Science. He is currently a full researcher from INRIA, working at the ENS Lyon. He is the author of one book, more than 25 papers published in international journals, and more than 35 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for clusters and grids. He is a member of the IEEE.