DOI 10.1007/s10951-008-0078-4

Minimizing the stretch when scheduling flows of divisible requests

Arnaud Legrand · Alan Su · Frédéric Vivien

Received: 16 November 2006 / Accepted: 23 June 2008 / Published online: 23 August 2008

Abstract In this paper, we consider the problem of scheduling distributed biological sequence comparison applications. This problem lies in the divisible load framework with negligible communication costs. Thus far, very few results have been proposed for this model. We discuss and select relevant metrics for this framework: namely max-stretch and sum-stretch. We explain the relationship between our model and the preemptive single processor case, and we show how to extend algorithms that have been proposed in the literature for the single processor model to the divisible multi-processor problem domain. We recall known results on closely related problems, we show how to minimize the max-stretch on unrelated machines either in the divisible load model or with preemption, we derive new lower bounds on the competitive ratio of any online algorithm, we present new competitiveness results for existing algorithms, and we develop several new online heuristics. We also address the Pareto optimization of max-stretch. Then, we extensively study the performance of these algorithms and heuristics under realistic scenarios. Our study shows that all previously proposed guaranteed heuristics for max-stretch for the single processor model are inefficient in practice. In contrast, we show that our online algorithms based on linear programming are in practice near-optimal solutions for max-stretch. Our study also clearly suggests heuristics that are efficient for both metrics, although a combined optimization is in theory not possible in the general case.

Keywords Bioinformatics · Heterogeneous computing · Scheduling · Divisible load · Linear programming · Stretch

1 Introduction

The problem of searching large-scale genomic and proteomic sequence databanks is an increasingly important bioinformatics problem. The results we present in this paper concern the deployment of such applications in heterogeneous parallel computing environments. In the genomic sequence comparison scenario, the presence of the required databank on a particular node is the sole factor that constrains task placement decisions. This application is thus part of a larger class of applications, in which each task in the application workload exhibits an "affinity" for particular nodes of the targeted computational platform. In this context, task affinities are determined by location and replication of the sequence databanks in the distributed platform.

Numerous efforts to parallelize biological sequence comparison applications have been realized (e.g., Braun et al. 2001; Darling et al. 2003; Miller et al. 1991). These efforts are facilitated by the fact that such biological sequence comparison algorithms are typically computationally intensive, embarrassingly parallel workloads. In the scheduling literature, this computational model is effectively a *divisible workload scheduling* problem (Bharadwaj et al. 1996; Blazewicz et al. 2007) with negligible communication overheads. The work presented in this paper concerns this application model, particularly in the context of *online scheduling* (i.e., in which the scheduler has no knowledge of any job

A. Legrand
CNRS, France Université de Grenoble, France LIG, UMR 5217,
CNRS–Grenoble INP–INRIA–UJF–UPMF, Grenoble, France

A. Su
Google Inc., Cambridge, USA

F. Vivien (✉)
INRIA, France Université de Lyon, France LIP, UMR 5668,
ENS-Lyon–CNRS–INRIA–UCBL, Lyon, France
e-mail: frederic.vivien@ens-lyon.fr

in the workload in advance of its release date). Thus far, this specific problem has not been considered in the scheduling literature.

Aside from divisibility, the main difference with classical scheduling problems lies in the fact that the platforms we target are shared by many users. Consequently, we need to ensure a certain degree of fairness between the different users and requests. Defining a fair objective that accounts for the various job characteristics (release date, processing time) is thus the first difficulty to overcome. After having presented our motivating application and our framework in Sect. 2, we review various classical metrics in Sect. 3 and conclude that the *stretch* of a job is an appropriate basis for evaluation. As a consequence, we mainly focus on the max-stretch and sum-stretch metrics. To have a good background on related objectives functions and results, in Sect. 4 we focus on the max-flow and sum-flow metrics. Then in Sect. 5 we study sum-stretch optimization, in Sect. 6 offline max-stretch optimization, and in Sect. 7 Pareto offline optimization of max-stretch. Building on the previous sections, we focus in Sect. 8 on the online optimization of max-stretch. This paper contains no section devoted to the related work as the related work will be discussed throughout this article. However, we summarize in the conclusions the known and new results on complexity. Finally, we present in Sect. 9 an experimental evaluation of the different solutions proposed, and we conclude in Sect. 10.

The main contributions of this work are:

- OFFLINE SUM-FLOW AND SUM-STRETCH. We show that sum-flow minimization is NP-complete on unrelated machines under the divisible load model ($\langle R|r_j, \text{div}|\sum F_j\rangle$ is NP-complete). We also show that sum-stretch minimization is NP-complete on one machine without preemption and also on unrelated machines under the divisible load model ($\langle 1|r_j|\sum S_j\rangle$ and $\langle R|r_j, \text{div}|\sum S_j\rangle$ are NP-complete).
- OFFLINE MAX WEIGHTED FLOW. We present polynomial-time algorithms to solve the minimization of max weighted flow, offline, on unrelated machines, in the divisible load model and in the preemptive model: $\langle R|r_j; \text{div}|\max w_j F_j\rangle$ and $\langle R|r_j; \text{pmtn}|\max w_j F_j\rangle$ are polynomial.

  We also propose heuristics to solve the offline Pareto minimization of max weighted flow, either on one machine or on unrelated machines. We present some cases in which these heuristics are optimal and we prove that the offline Pareto minimization of max-flow on unrelated machines is NP-complete.
- ONLINE SUM-STRETCH AND MAX-STRETCH. We show that *First come, first served* (FCFS) is Δ^2-competitive for sum-stretch minimization and Δ-competitive for max-stretch, where Δ denotes the ratio of the sizes of the largest and shortest jobs submitted to the system. We also prove that no online algorithm has simultaneously better competitive ratios for these two metrics.

  We show that no online algorithm has a competitive ratio less than or equal to 1.19484 for the minimization of sum-stretch, or less than or equal to $\frac{1}{2}\Delta^{\sqrt{2}-1}$ for the minimization of max-stretch. (The previous known bounds were, respectively, 1.036 and $\frac{1}{2}\Delta^{\frac{1}{3}}$.)

  For minimizing the sum-stretch on one machine with preemption, we show that Smith's ratio rule—which is then equivalent to *shortest processing time*—is not a competitive algorithm and that *shortest weighted remaining processing time* is at best 2-competitive.

  Finally, we propose new heuristics for the online optimization of max-stretch. Through extensive simulations we compare them with solutions found in the literature and we show their very good performance.

All the details and proofs missing from this article can be found in its companion research report (Legrand et al. 2008).

2 Motivating application and framework

2.1 Motivating application

The only purpose of this section is to present the application that originally motivated this work, the GriPPS (Blanchet et al. 2000; GriPPS webpage 2005) protein comparison application. The GriPPS framework is based on large databases of information about proteins; each protein is represented by a string of characters denoting the sequence of amino acids of which it is composed. Biologists need to search such sequence databases for specific patterns that indicate biologically significant structures. The GriPPS software enables such queries in grid environments, where the data may be replicated across a distributed heterogeneous computing platform.

As a matter of fact, there seems to be two common usages in protein comparison applications. In the first case, a biologist working on a set of proteins builds a pattern to search for similar sequences on the servers (this is the case for the GriPPS framework). In the second case, canonical patterns are known and should be used for comparison with daily updates of the databanks. This is the only case we are aware of where a very large set of motifs is sent to all databanks. This is, however, a typical background process, whereas the first case is a typical online problem as many biologists concurrently use the servers. Therefore, in this first case, the motifs are very small and communication cost they incur can really be neglected. To develop a suitable application model for the GriPPS application scenario, we performed a series of experiments to analyze the fundamental properties of the sequence comparison algorithms used in this code. Here we report on the conclusions of this study whose details can be found in Legrand et al. (2005, 2004).

From our modeling perspective, the critical components of the GriPPS application are:

1. *Protein databanks*: the reference databases of amino acid sequences, located at fixed locations in a distributed heterogeneous computing platform.
2. *Motifs*: compact representations of amino acid patterns that are biologically important and serve as user input to the application.
3. *Sequence comparison servers*: computational processes co-located with protein databanks that accept as input sets of motifs and return as output all matching entries in any subset of a particular databank.

The main characteristics of the GriPPS application are:

1. *Negligible communication costs*. A motif is a relatively compact representation of an amino acid pattern. Therefore, the communication overhead induced while sending a motif to any processor is negligible compared to the processing time of a comparison.
2. *Divisible loads*. The processing time required for sequence comparisons against a subset of a particular databank is linearly proportional to the size of the subset. This property allows us to distribute the processing of a request among many processors at the same time without additional cost.

   The GriPPS protein databank search application is therefore an example of a *linear divisible workload without communication costs*.

   In the classical scheduling literature, preemption is defined as the ability to suspend a job at any time and to resume it, possibly on another processor, at no cost. Our application implicitly falls in this category. Indeed, we can easily halt the processing of a request on a given processor and continue the pattern matching for the unprocessed part of the database on a different processor (as it only requires a negligible data transfer operation to move the pattern to the new location). From a theoretical perspective, divisible load without communication costs can be seen as a generalization of the *preemptive execution model* that allows for simultaneous execution of different parts of a same job on different machines.
3. *Uniform machines with restricted availabilities*. A set of jobs is uniform over a set of processors if the relative execution times of jobs over the set of processors does not depend on the nature of the jobs. More formally, for any job J_j, the time $p_{i,j}$ needed to process job J_j on processor i is equal to $W_j \cdot c_i$, where c_i describes the speed of processor i and W_j represents the size of J_j. Our experiments indicated a clear constant relationship between the computation time observed for a particular motif on a given machine, compared to the computation time measured on a reference machine for that same motif. This trend supports the hypothesis of uniformity. However, in practice a given databank may not be available on all sequence comparison servers. Our model essentially represents a *uniform machines with restricted availabilities* scheduling problem, which is a specific instance of the more general *unrelated machines* scheduling problem.

2.2 Framework and notations

Formally, an instance of our problem is defined by n jobs, $J_1, \ldots, J_n$, and m machines (or processors), $M_1, \ldots, M_m$. The job J_j arrives in the system at time r_j (expressed in seconds), which is its release date; we suppose that jobs are numbered by increasing release dates.

The value $p_{i,j}$ denotes the amount of time it would take for machine M_i to process job J_j. Note that $p_{i,j}$ can be infinite if the job J_j cannot be executed on the machine M_i, e.g., for our motivating application, if job J_j requires a databank that is not present on the machine M_i. Finally, each job is assigned a *weight* or *priority* w_j.

As we have seen, for the particular case of our motivating application, we could replace the unrelated times $p_{i,j}$ by the expression $W_j \cdot c_i$, where W_j denotes the size (in Mflop) of the job J_j and c_i denotes the computational capacity of machine M_i (in second·Mflop^{-1}). To maintain correctness for the biological sequence comparison application, we separately maintain a list of databanks present at each machine and enforce the constraint that a job J_j may only be executed on a machine that has a copy of all data upon which job J_j depends. However, since the theoretical results we present do not rely on these restrictions, we retain the more general scheduling problem formulation that is, we address the unrelated machines framework in this article. As a consequence, all the values we consider in this article are nonnegative rational numbers (except the previously mentioned case in which $p_{i,j}$ is infinite if J_j cannot be processed on M_i).

The time at which job J_j is completed is denoted as C_j. Then, the *flow time* of the job J_j, defined as $F_j = C_j - r_j$, is essentially the time the job spends in the system.

Due to the divisible load model, each job may be divided into an arbitrary number of sub-jobs, of any size. Furthermore, each sub-job may be executed on any machine at which the data dependencies of the job are satisfied. Thus, at a given moment, many different machines may be processing the same job (with a master scheduler ensuring that these machines are working on *different* parts of the job). Therefore, if we denote by $\alpha_{i,j}$ the fraction of job J_j processed on M_i, we enforce the following property to ensure each job is fully executed: $\forall j, \sum_i \alpha_{i,j} = 1$.

When a size W_j can be defined for each job J_j—e.g., in the single processor case—we denote by Δ the ratio of the sizes of the largest and shortest jobs submitted to the system: $\Delta = \frac{\max_j W_j}{\min_j W_j}$.

2.3 Relationships with the single processor case with preemption

We first prove that any schedule in the uniform machines model with divisibility has a canonical corresponding schedule in the single processor model with preemption. This is especially important as many interesting results in the scheduling literature only hold for the preemptive computation model (denoted 'pmtn').

Lemma 1 *For any platform $M_1, \ldots, M_m$ composed of uniform processors, i.e., such that for any job J_j, $p_{i,j} = W_j \cdot c_i$, one can define a platform made of a single processor $\widetilde{M}$ with $\widetilde{c} = 1/\sum_i \frac{1}{c_i}$, such that:*

For any divisible schedule of $J_1, \ldots, J_n$ on $\{M_1, \ldots, M_m\}$ there exists a preemptive schedule of $J_1, \ldots, J_n$ on $\widetilde{M}$ with smaller or equal completion times.

Conversely, for any preemptive schedule of $J_1, \ldots, J_n$ on $\widetilde{M}$ there exists a divisible schedule of $\{M_1, \ldots, M_m\}$ with equal completion times.

Figure 1 illustrates the underlying idea (see research report, Legrand et al. 2008, for details). The reverse transformation simply processes jobs sequentially, distributing each job's work across all processors. As a consequence, any complexity result for the preemptive single processor model also holds for the uniform divisible model. Thus, throughout this article, in addition to addressing the multi-processor case, we will also closely examine the single processor case.

Unfortunately, this line of reasoning is no longer valid when the computational platform exhibits restricted availability, as defined in Sect. 2. In the single processor case, a schedule can be seen as a priority list of the jobs (see the article of Bender et al. 2004, for example). For this reason, whenever we will present heuristics for the uniprocessor case they will follow the same basic approach: maintain a priority list of the jobs and at any moment, execute the one with the highest priority. In the multi-processor case with restricted availability, an additional scheduling dimension must be resolved: the spatial distribution of each job.

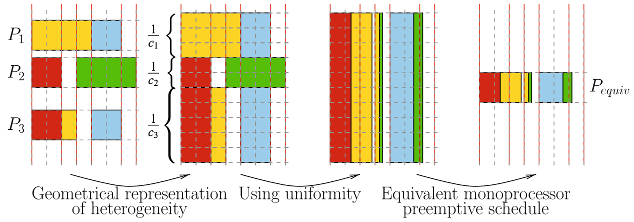

P_1
P_2
P_3
$\frac{1}{c_1}$
$\frac{1}{c_2}$
$\frac{1}{c_3}$
P_{equiv}
Geometrical representation of heterogeneity
Using uniformity
Equivalent monoprocessor preemptive schedule

Fig. 1 Geometrical transformation of a divisible uniform problem into a preemptive single processor problem

The example in Fig. 2 explains the difficulty of this last problem. In the uniform situation, it is always beneficial to fully distribute work across all available resources: each job's completion time in situation B is strictly better than the corresponding job's completion time in situation A. However, introducing restricted availability confounds this process. Consider a case in which tasks may be limited in their ability to utilize some subset of the platform's resources (e.g., their requisite data are not present throughout the platform). In situation C of Fig. 2, one task is subject to restricted availability: the P_2 computational resource is not able to service this task. Deciding between various scheduling options in this scenario is non-trivial in the general case (for example, schedule A has a better max-flow than schedule C, but schedule C has a better max-stretch than schedule A), so we apply Algorithm 1's simple rule to build a schedule for general platforms from single processor heuristics.

Another important characteristic of our problem is that we target a platform shared by many users. As a consequence, we need to ensure a certain degree of fairness between the different requests. Given a set of requests, how should we share resources amongst the different requests? The next section examines objective functions that are well-suited to achieve this notion of fairness.

3 Objective functions

We first recall several common objective functions in the scheduling literature and highlight those that are most rel-

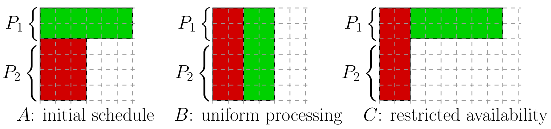


Fig. 2 Illustrating the difference between the uniform model and the restricted availability model

Algorithm 1 Converting a single-processor schedule to a divisible one with restricted availability

```
while some processors are idle do
    Select the job with the highest priority and distribute its processing on all available processors
    that are capable of processing it.
```

evant to our work (Sect. 3.1). Then, we show that the optimization of certain objectives are mutually exclusive (Sect. 3.2).

3.1 Looking for a fair objective function

The most common objective function in the parallel scheduling literature is the *makespan*: the maximum of the job termination times, or $\max_j C_j$. Makespan minimization is conceptually a system-centric approach, seeking to ensure efficient platform utilization. Makespan minimization is meaningful when there is only one user and when all jobs are submitted simultaneously. However, individual users sharing a system are typically more interested in job-centric metrics, such as *job flow time* (also called *response time*): the time an individual job spends in the system. Optimizing the average (or total) flow time, $\sum_j F_j$, suffers from the limitation that starvation is possible, i.e., some jobs may be delayed to an unbounded extent (Bender et al. 1998). By contrast, minimization of the maximum flow time, $\max_j F_j$, does not suffer from this limitation, but it tends to favor long jobs to the detriment of short ones. To overcome this problem, one common approach (Chekuri and Khanna 2002) focuses on the *weighted* flow time, using job weights to offset the bias against short jobs. *Sum weighted flow* and *maximum weighted flow* metrics can then be analogously defined. Note, however, that the starvation problem identified for sum-flow minimization is inherent to all sum-based objectives, so the sum weighted flow suffers from the same weakness. The *stretch* is a particular case of weighted flow, in which a job's weight is inversely proportional to its size: $w_j = 1/W_j$ (Bender et al. 1998). On a single processor, the stretch of a job can be seen as the slowdown it experiences when the system is loaded. In a network context, the stretch can be seen as the inverse of the overall bandwidth allocated to a given transfer (i.e., the amount of data to transfer divided by the overall time needed to complete the transfer). However, this kind of definition does not account for the affinity of some tasks with some particular machines (e.g., the scarcity of a particular database). That is why we think a slightly different definition should be used in an unrelated machines context. The stretch is originally defined to represent the slowdown a job experiences when the system is loaded. In the remaining of this article, we will thus define the stretch as a particular case of weighted flow, in which a job's weight is inversely proportional to its processing time when the system is empty: $w_j = \sum_i \frac{1}{p_{i,j}}$ in our divisible load model. This definition matches the previous one in a single processor context and is thus a reasonably fair measure of the level of service provided to an individual job. It is more relevant than the flow in a system with highly variable job sizes. Consequently, this article focuses mainly on the sum-stretch ($\sum S_j$) and the max-stretch ($\max S_j$) metrics.

3.2 Simultaneous online optimization of sum-stretch and max-stretch is impossible

We prove that simultaneously optimizing the objectives we have defined earlier (sum-stretch and max-stretch) may be impossible in certain situations.[1] In this section, we only consider the single processor case.

Theorem 1 *Consider any online algorithm which has a competitive ratio of $\rho(\Delta)$ for the sum-stretch. We assume that this competitive ratio is not trivial, i.e., that $\rho(\Delta) < \Delta^2$. Then, there exists for this algorithm a sequence of jobs that leads to starvation, and thus for which the obtained max-stretch is arbitrarily greater than the optimal max-stretch.*

Note that the currently best known online algorithm for sum-stretch is 2-competitive (see Sect. 5.3). Using the exact same construction, we can show that for any online algorithm which has a non-trivial competitive ratio of $\rho(\Delta) < \Delta$ for the sum-flow, there exists a sequence of jobs leading to starvation and where the obtained max-flow is arbitrarily greater than the optimal one.

We must comment on our assumption about *non-trivial competitive ratios*. This comes from the fact that ignoring job sizes leads on a single processor to a Δ^2-competitive online algorithm for sum-stretch and Δ-competitive online algorithm for sum-flow:

Theorem 2 FCFS *is*:

- Δ^2*-competitive for the online minimization of sum-stretch*
- Δ*-competitive for the online minimization of max-stretch*
- Δ*-competitive for the online minimization of sum-flow, and*
- *optimal for the online minimization of max-flow* (*classical result*, *see* Bender et al. 1998, *for example*).

Proofs and details can be found in the research report corresponding to this article (see Legrand et al. 2008). We now prove Theorem 1.

Proof Let us consider the case of an online algorithm for the sum-stretch optimization problem that achieves a competitive ratio of $\rho(\Delta) < \Delta^2$. We arbitrarily take a value for $\Delta > 1$. Then, there exists $\varepsilon > 0$, such that $\rho(\Delta) < \Delta^2 - \varepsilon$. Finally, let α be any integer such that $\frac{1+\alpha\Delta}{1+\frac{\alpha}{\Delta}} > \Delta^2 - \frac{\varepsilon}{2}$ (note that this is the case for any value of α which is large enough).

[1] Note that the following two theorems have been incorrectly stated in Legrand et al. (2006).

At date 0 arrives α jobs, $J_1, \dots, J_\alpha$, of size Δ. Let k be any integer. Then, at any time unit t, $0 \leq t \leq k-1$, arrives a job $J_{\alpha+t+1}$ of size 1.

A possible schedule would be to process each of the k jobs of size 1 at its release date, and to wait for the completion of the last of these jobs before processing the jobs $J_1, \dots, J_\alpha$. The sum-stretch would then be $k \times 1 + \frac{k+\Delta}{\Delta} + \dots + \frac{k+\alpha\Delta}{\Delta} = \frac{\alpha(\alpha+1)}{2} + k(1+\frac{\alpha}{\Delta})$ and the max-stretch would be $\alpha + \frac{k}{\Delta}$. Even if it is not optimal for either criterion, we can still use it as an upper-bound.

In fact, with our hypotheses, the online algorithm cannot complete the execution of all the jobs $J_1, \dots, J_\alpha$ as long as there are jobs of size 1 arriving at each time unit. Otherwise, suppose that at some date t_1, jobs $J_1, \dots, J_\alpha$ have all been completed. Then, a certain number k_1 of unit-size jobs were completed before time t_1. The scenario which minimizes the sum-stretch under these constraints is to schedule first the k_1 jobs $J_{\alpha+1}, \dots, J_{\alpha+k_1}$ at their release dates, then to schedule $J_1, \dots, J_\alpha$, and then the remaining $k - k_1$ jobs of size 1. The sum-stretch of the actual schedule cannot therefore be smaller than the sum-stretch of this schedule, which is equal to:

$$k_1 \times 1 + \frac{k_1+\Delta}{\Delta} + \dots + \frac{k_1+\alpha\Delta}{\Delta} + (k-k_1)(1+\alpha\Delta)$$
$$= \left(\frac{\alpha(\alpha+1)}{2} + \frac{\alpha k_1}{\Delta}\right) + k_1 + (k-k_1)(1+\alpha\Delta).$$

However, as, by hypothesis, we consider a $\rho(\Delta)$-competitive algorithm, the obtained schedule must at most be $\rho(\Delta)$ times the optimal schedule. This implies that:

$$\left(\frac{\alpha(\alpha+1)}{2} + \frac{\alpha k_1}{\Delta}\right) + k_1 + (k-k_1)(1+\alpha\Delta)$$
$$\leq \rho(\Delta)\left(\frac{\alpha(\alpha+1)}{2} + k\left(1+\frac{\alpha}{\Delta}\right)\right)$$
$$\iff \quad -\alpha\Delta k_1 + \frac{\alpha(\alpha+1)}{2}\big(1-\rho(\Delta)\big) + \frac{\alpha k_1}{\Delta}$$
$$\leq k\left(\rho(\Delta)\left(1+\frac{\alpha}{\Delta}\right) - (1+\alpha\Delta)\right).$$

Once the approximation algorithm has completed the execution of the jobs $J_1, \dots, J_\alpha$ we can keep sending unit-size jobs for k to become as large as we wish. Therefore, for the inequality not to be violated, we must have $\rho(\Delta)(1+\frac{\alpha}{\Delta}) - (1+\alpha\Delta) \geq 0$. However, we have by hypothesis $\rho(\Delta) < \Delta^2 - \varepsilon$. Therefore, we must have $\Delta^2 - \varepsilon > \frac{1+\alpha\Delta}{1+\frac{\alpha}{\Delta}}$, which contradicts the definition of α. Therefore, the only possible behavior for the approximation algorithm is to delay the execution of at least one of the jobs $J_1, \dots, J_\alpha$ until after the end of the arrival of the unit-size jobs, whatever the number of these jobs. This leads to the starvation of at least one of these jobs. Furthermore, the ratio of the obtained max-stretch to the optimal one is $\frac{\alpha+\frac{k}{\Delta}}{1+\alpha\Delta} = \frac{\alpha\Delta+k}{\Delta(\alpha\Delta+a)}$, which may be arbitrarily large. □

Intuitively, algorithms targeting max-based metrics ensure that no job is *left behind*. Such an algorithm is thus extremely “fair” in the sense that cost (in our context the weighted flow or the stretch of each job) is made as equal as possible for everybody. Sum-based metrics tend to optimize instead the *utilization* of the platform. The previous theorem establishes that these two objectives can be in opposition on particular instances. As a consequence, it should be noted that any algorithm optimizing a sum-based metric has the particularly undesirable property of potential starvation. This observation, combined with the fact that the stretch is more relevant than the flow in a system with highly variable job sizes, motivates max-stretch as the metric of choice in designing scheduling algorithms in the GriPPS setting.

4 Flow optimization

On a single processor, the max-flow is optimized by FCFS (see Bender et al. 1998, for example). Using the remarks of Sect. 2.3, we can thus easily derive an online optimal algorithm for $\langle Q|r_j;\text{div}|F_{\max}\rangle$. We will see in Sect. 6 that $\langle R|r_j;\text{div}|\max w_j F_j\rangle$ can be solved in polynomial time using linear programming techniques.

Regarding sum-flow, it was proved by Baker (1974), using exchange arguments, that SRPT (shortest remaining processing time first) is optimal for the $\langle 1|r_j;\text{pmtn}|\sum C_j\rangle$ problem. It is thus also optimal for $\langle 1|r_j;\text{pmtn}|\sum F_j\rangle$ and, using the remarks of Sect. 2.3, we can easily derive an online optimal algorithm for $\langle Q|r_j;\text{div}|\sum F_j\rangle$. We will, however, see in this section that under the *uniform machines with restricted availabilities* model, this problem is much harder.

Many of the reduction we propose in this article rely on the following strongly NP-hard problem (Garey and Johnson 1991):

Definition 1 (3-Dimensional Matching (3DM) problem) Given three sets $U = \{u_1, \dots, u_m\}$, $V = \{v_1, \dots, v_m\}$, and $W = \{w_1, \dots, w_m\}$, and a subset $S \subset U \times V \times W$ of size $n \geq m$, does S contain a perfect matching, that is, a set $S' \subseteq S$ of cardinality m that covers every element in $U \cup V \cup W$?

Theorem 3 *The scheduling problem* $\langle R|r_j, \text{div}|\sum F_j\rangle$ *is NP-complete.*

The reduction is made from *3-Dimensional Matching* and uses the same idea as Sitters (2005) used to prove the strong

NP-hardness of $\langle R|\text{pmtn}|\sum C_j\rangle$. It should be noted that, in the reduction we use, the machines are uniform machines with restricted availabilities. We refer the reader to the extended version of this document (Legrand et al. 2008) where the whole proof can be found.

5 Sum-stretch optimization

In this section, we give various results regarding sum-stretch optimization. In Sect. 5.1, we establish the complexity of this problem in our framework. In the remaining sections, we focus on the one processor setting and study the competitiveness of "classical" heuristics.

5.1 Complexity of the offline problem

In the general case, without preemption and divisibility, minimizing the sum-stretch is an NP-complete problem:

Theorem 4 *The scheduling problem $\langle 1|r_j|\sum S_j\rangle$ is NP-complete.*

This NP-completeness result is proved by reduction from of Partition. A complete proof can be found in the research report corresponding to this article (see Legrand et al. 2008).

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. In the framework with preemption, Bender et al. (2004) present a Polynomial Time Approximation Scheme (PTAS) for minimizing the sum-stretch with preemption. Chekuri and Khanna (2002) present an approximation scheme for the more general sum weighted flow minimization problem. As these approximation schemes cannot be extended to work in an online setting, we will not discuss them further.

Moving to the divisible load framework, we can easily say that the complexity of $\langle Q|r_j;\text{div}|\sum S_j\rangle$ is open (using the remarks of Sect. 2.3). The minimization of the sum-stretch is, however, NP-complete on unrelated machines:

Theorem 5 *The scheduling problem $\langle R|r_j,\text{div}|\sum S_j\rangle$ is NP-complete.*

The reduction is also made from *3-Dimensional Matching* and also uses the same idea as Sitters (2005) used to prove the strong NP-hardness of $\langle R|\text{pmtn}|\sum C_j\rangle$. In the reduction, the machines are thus also uniform machines with restricted availabilities. A complete proof can be found in the research report corresponding to this article (see Legrand et al. 2008).

5.2 Lower bound on the competitiveness of online algorithms

Muthukrishnan et al. (1999) propose an optimal online algorithm when there are only two job sizes. Mainly, they prove that there is no optimal online algorithm for the sum-stretch minimization problem when there are three or more distinct job sizes. Furthermore, they give a lower bound of 1.036 on the competitive ratio of any online algorithm. The following theorem improves this bound:

Theorem 6 (Legrand et al. 2006) *No online algorithm minimizing the sum-stretch with preemption on a single processor has a competitive ratio less than or equal to* 1.19484.

Proof Here we just present the adversary used to prove this lower bound. A complete proof can be found in the research report corresponding to this article (Legrand et al. 2008). The adversary behavior is defined by the following parameters: $\alpha = 1.93716$, $\beta = 1.29941$, $n = \frac{1}{\varepsilon} \approx 2.69598$, $k = 10^{12}$, and $\varepsilon = 0.370923$. Our adversary proceeds as follows:

1. At time $r_0 = 0$ we send a job J_0 of size $p_0 = \alpha\beta n$.
2. At time $r_1 = \alpha\beta n - \varepsilon$ we send a job J_1 of size $p_1 = \beta n$.
3. We consider the system at time $\alpha\beta n + \beta n - \varepsilon$, and whether the execution of J_0 has been completed at that time.
   (a) If the execution of J_0 has not yet been completed, we do not send any more jobs.
   (b) Otherwise, at time $r_2 = \alpha\beta n + \beta n - \varepsilon$ we send a job J_2 of size $p_2 = n$.
   We consider the system at time $\alpha\beta n + \beta n + n - \varepsilon$, and whether the execution of J_1 has been completed at that time.
      i. If the execution of J_1 has not yet been completed, we do not send any more jobs.
      ii. Otherwise, at time $r_3 = \alpha\beta n + \beta n + n - \varepsilon$ we send a job J_3 of size $p_3 = n$.
      We consider the system at time $\alpha\beta n + \beta n + 2n - \varepsilon$, and whether the execution of J_2 has been completed at that time.
         A. If the execution of J_2 has not yet been completed, we do not send any more jobs.
         B. Otherwise, we send to the system a series of k unit-size jobs, the inter-arrival time of these jobs being equal to their size: at time $r_{3+j} = \alpha\beta n + \beta n + 2n - \varepsilon + (j-1)$ we send the job J_{3+j} of size $p_{3+j} = 1$, for $1 \leq j \leq k$.

The parameters are chosen so that the optimal completion order of the jobs is J_1, J_2, J_3, $J_{3+1}, \ldots, J_{3+k}$, J_0. □

5.3 Shortest processing time rules: SRPT, SWPT, SWRPT

In the previous section, we have recalled that *shortest remaining processing time* (SRPT) is optimal for minimizing the sum-flow. When SRPT takes a scheduling decision, it only considers the *remaining* processing time of a job, and not its *original* processing time. Therefore, from the point of view of the sum-stretch minimization, SRPT does not take into account the *weight* of the jobs in the objective function. Nevertheless, Muthukrishnan, Rajaraman, Shaheen, and Gehrke have shown (Muthukrishnan et al. 1999) that SRPT is 2-competitive for sum-stretch.

Another well studied algorithm is the Smith's ratio rule (Smith 1956) also known as *shortest weighted processing time* (SWPT). This is a preemptive list scheduling where the available jobs are executed in increasing value of the ratio $\frac{p_j}{w_j}$. Whatever the weights, SWPT is 2-competitive (Schulz and Skutella 2002) for the minimization of the sum of weighted completion times ($\sum w_j C_j$). Note that a ρ-competitive algorithm for the sum weighted flow minimization ($\sum w_j(C_j - r_j)$) is ρ-competitive for the sum weighted completion time ($\sum w_j C_j$). However, the reverse is not true: a guarantee on the sum weighted completion time ($\sum w_j C_j$) does not induce any guarantee on the sum weighted flow ($\sum w_j(C_j - r_j)$). Therefore, the previous ratio on the minimization of the sum of weighted completion times gives us no result on the efficiency of SWPT for the minimization of the sum-stretch. Furthermore, we can even prove that SWPT is not an approximation algorithm for minimizing the sum-stretch. Indeed, SWPT schedules the available jobs by increasing values of $\frac{1}{p_j^2}$ and has thus exactly the same behavior as the *shortest processing time* first heuristic (SPT). The following theorem states that SPT (and thus SWPT) is not a competitive algorithm for minimizing the sum-stretch.

Theorem 7 (Legrand et al. 2006) *For any value $\rho > 1$, there is an instance on which the sum-stretch realized by* SPT *is at least ρ times the optimal. Furthermore, we can impose that in this instance Δ, the ratio of the sizes of the largest and shortest jobs submitted to the system, is equal to* 2.

Proof Without loss of generality, we assume that ρ is a strictly positive integer. Then, the problematic instance is made of $4\rho + 1$ jobs where job J_k, $0 \le k \le 4\rho$, is defined by: $r_k = 8\rho k - \frac{k(k+1)}{2}$ and $p_k = 8\rho - k$. A complete proof can be found in the research report corresponding to this article (see Legrand et al. 2008). □

The weakness of the SWPT heuristics is obviously that it does not take into account the remaining processing times: it may preempt a job when it is almost completed. To address the weaknesses of both SRPT and SWPT, one might consider a heuristic that takes into account both the original and the remaining processing times of the jobs. This is what the *shortest weighted remaining processing time* heuristic (SWRPT) does. In the framework of sum-stretch minimization, at any time t, SWRPT schedules the job J_j which minimizes $p_j \rho_t(j)$. Muthukrishnan et al. (1999) prove that SWRPT is actually optimal when there are only two job sizes.

Neither of the proofs of competitiveness of SRPT or SWPT can be extended to SWRPT. SWRPT has apparently been studied by Megow (2002), but only in the scope of the sum weighted completion time. So far, there is no guarantee on the efficiency of SWRPT for sum-stretch minimization. Intuitively, we would think that SWRPT is more efficient than SRPT for the sum-stretch minimization. However, the following theorem shows that the worst case for SWRPT for the sum-stretch minimization is no better than that of SRPT.

Theorem 8 (Legrand et al. 2006) *For any real ε, $1 > \varepsilon > 0$, there exists an instance such that* SWRPT *is not $(2-\varepsilon)$-competitive for the minimization of the sum-stretch.*

Proof Here we just present the construction used to prove this lower bound on the competitive ratio of SWRPT. A complete proof can be found in the research report corresponding to this article (see Legrand et al. 2008). The problematic instance is composed of two sequences of jobs. In the first sequence, the jobs are of decreasing sizes, the size of a job being the square root of the size of its immediate predecessor. In the second sequence, all the jobs are of unit-size. Each job arrives at a date equal to the release date of its predecessor plus the execution time of this predecessor, except for the second and third jobs which arrive at dates critical for SWRPT.

Let $\alpha = 1 - \frac{\varepsilon}{3}$, $n = \lceil \log_2(\log_2 \frac{3(1+\alpha)}{\varepsilon}) \rceil$, and $k = \lceil -\log_2(-\log_2 \alpha) \rceil$. Let l be an integer. Then, we formally build the instance $\mathcal{J}$ as follows:

1. Job J_0 arrives at time $r_0 = 0$ and is of size $p_0 = 2^{2^n}$.
2. Job J_1 arrives at time $r_1 = 2^{2^n} - 2^{2^{n-2}}$ and is of size $p_1 = 2^{2^{n-1}}$.
3. Job J_2 arrives at time $r_2 = r_1 + 2^{2^{n-1}} - \alpha$ and is of size $p_2 = 2^{2^{n-2}}$.
4. Job J_j, for $3 \le j \le n$, arrives at time $r_j = r_{j-1} + p_{j-1}$ and is of size $p_j = 2^{2^{n-j}}$.
5. Job J_{n+j}, for $1 \le j \le k$, is of size $p_{n+j} = 2^{2^{-j}}$ and arrives at time $r_{n+j} = r_{n+j-1} + p_{n+j-1}$.
6. Job J_{n+k+j}, for $1 \le j \le l$, is of size $p_{n+k+j} = 1$ and arrives at time $r_{n+k+j} = r_{n+k+j-1} + p_{n+k+j-1}$.

Then, we evaluate the sum-stretch realized by both SWRPT and SRPT and we show that, if l is large enough, the sum-stretch realized by SWRPT is ($\mathcal{R} \geq 2 - \varepsilon$)-times that realized by SRPT. This proves the result as the optimal sum-stretch is no greater than that of SRPT. □

6 Offline max-stretch optimization

Bender et al. (1998) have shown that the problem of max-stretch minimization on one machine *without* preemption, i.e., problem $\langle 1|r_j|S_{\max}\rangle$, cannot be approximated within a factor $O(n^{1-\varepsilon})$ for arbitrarily small $\varepsilon > 0$, unless P = NP. In this section, we show that if we allow either divisible loads or preemptions, we are able to minimize the maximum weighted flow in polynomial time even on unrelated machines.

In Sect. 6.1, we state the relationship between minimization of the maximum weighted flow problem and deadline scheduling. Then we present a solution to maximum weighted flow minimization in the divisible load framework, on unrelated machines. By adapting some of these techniques, we then describe a solution to the minimization of the maximum weighted flow when preemption (but not load divisibility) is allowed, once again on unrelated machines. These results are given in Sect. 6.2.

It should be noted that, prior to our work, at least two solutions were known for minimizing the max-stretch on one machine with preemption. Baker et al. (1983) presented an $O(n^2)$ algorithm to solve an even more general problem: $\langle 1|\text{pmtn, prec}, r_j|f_{\max}\rangle$ (where $f_{\max}$ is the maximum of the costs of the jobs and the cost of a job is a non-decreasing function of its completion time). This algorithm determines the job of least priority and then iterates. Another solution using network flow maximization techniques was known.[2] In our divisible load framework, we do not know how to extend this flow maximization technique to solve the case of uniform machines with restricted availabilities, much less the more general case of unrelated processors. Nevertheless, for the sake of completeness, we recall this solution in Sect. 6.1.4. Finally, let us mention that the article (Lawler and Labetoulle 1978) contains explicitly, or implicitly, most (if not all) of the techniques used in this section. We nevertheless fully expose the max-stretch minimization algorithm as these techniques may not be so widely known[3] and as a good knowledge of the algorithm is necessary to understand the content of Sects. 7 and 8.

6.1 Minimizing the maximum weighted flow in the divisible model

6.1.1 *Max weighted flow minimization and deadline scheduling*

Let us assume that we are looking for a schedule $\mathcal{S}$ under which the maximum weighted flow is less than or equal to some objective value $\mathcal{F}$. The weighted flow of any job J_j is equal to $w_j(C_j - r_j)$. Then, we should have:

$$\begin{aligned} &\max_{1\leq j\leq n} w_j(C_j - r_j) \leq \mathcal{F} \\ &\iff \quad \forall j \in [1;n], \quad w_j(C_j - r_j) \leq \mathcal{F} \\ &\iff \quad \forall j \in [1;n], \quad C_j \leq r_j + \mathcal{F}/w_j. \end{aligned}$$

Thus, the execution of J_j must be completed before time $d_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ for schedule $\mathcal{S}$ to satisfy the bound $\mathcal{F}$ on the maximum weighted flow. Therefore, looking for a schedule which satisfies a given upper bound on the maximum weighted flow is equivalent to an instance of the deadline scheduling problem. We now show how to solve such a deadline scheduling problem in the divisible load framework.

In *deadline scheduling*, each job J_j has not only a release date r_j but also a deadline d_j. The problem is then to find a schedule such that each job J_j is executed within its executable time interval $[r_j, d_j]$. We consider the set of all job release dates and deadlines: $\{r_1, \ldots, r_n, d_1, \ldots, d_n\}$. We define an *epochal time* as a time value at which one or more points in this set occur; there are between 2 (when all jobs are released at the same date and have the same deadline) and $2n$ (when all job release dates and deadlines are distinct) such values. When ordered in absolute time, adjacent epochal times define a set of *time intervals*. We denote each time interval I_t by $I_t = [\inf I_t, \sup I_t[$. Finally, we denote by $\alpha_{i,j}^{(t)}$ the fraction of job J_j processed by machine M_i during the time interval I_t. In this framework, System (1) lists the constraints that should hold true in any valid schedule:

(a) *Release date*: job J_j cannot be processed before it is released (equation (1a)).
(b) *Deadline*: job J_j cannot be processed after its deadline (equation (1b)).
(c) *Resource usage*: during a time interval, a machine cannot be used longer than the duration of this time interval (equation (1c)).
(d) *Job completion*: each job must be processed to completion (equation (1d)).

[2] We do not know any reference to this technique which was presented to us by Michael Bender.

[3] For instance, Bender, Chakrabarti, and Muthukrishnan proved in Bender et al. (1998) the existence of a PTAS for a problem that is solved in polynomial-time in this section.

$$\left\{\begin{array}{ll}\text{(1a)} & \forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \;\Rightarrow\; \alpha_{i,j}^{(t)} = 0;\\[2mm] \text{(1b)} & \forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \;\Rightarrow\; \alpha_{i,j}^{(t)} = 0;\\[2mm] \text{(1c)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leq \sup I_t - \inf I_t;\\[2mm] \text{(1d)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1.\end{array}\right. \qquad (1)$$

Lemma 2 *System* (1) *has a solution if, and only if, there exists a solution to the deadline scheduling problem.*

System (1) can be solved in polynomial time by any linear solver system as all its variables are rational. Building a valid schedule from any solution of system (1) is straightforward as for any time interval I_t, and on any machine M_i, the job fractions $\alpha_{i,j}^{(t)}$ can be scheduled in any order.

One may think that by applying a binary search on possible values of the objective value $\mathcal{F}$, one would be able to find the optimal maximum weighted flow, and an optimal schedule. However, a binary search on rational values will not terminate. By setting a limit on the precision of the binary search, the number of process iterations is bounded, and the quality of the approximation can be guaranteed. However, as we now show, we can adapt our search to always find the optimal in polynomial time.

6.1.2 *Solving on a range*

So far we have used System (1) to check whether our problem has a solution whose maximum weighted flow is no greater than some objective value $\mathcal{F}$. We now show that we can use it to check whether our problem has a solution for some particular *range* of objective values. Later we show how to divide the whole search space into a polynomial number of search ranges.

First, let us suppose there exist two values $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1 < \mathcal{F}_2$, such that the relative order of the release dates and deadlines, $r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})$, when ordered in absolute time, is independent of the value of $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$. Then, on the objective interval $]\mathcal{F}_1, \mathcal{F}_2[$, as before, we define an epochal time as a time value at which one or more points in the set $\{r_1, \ldots, r_n, d_1(\mathcal{F}), \ldots, d_n(\mathcal{F})\}$ occurs. Note that an epochal time which corresponds to a deadline is no longer a constant but an affine function in $\mathcal{F}$. As previously, when ordered in absolute time, adjacent epochal times define a set of *time intervals*, that we denote by $I_1, \ldots, I_{n_{\text{int}}(\mathcal{F})}$. The durations of time intervals are now affine functions in $\mathcal{F}$. Using these new definitions and notations, we can solve our problem on the objective interval $[\mathcal{F}_1, \mathcal{F}_2]$ using System (1) with the additional constraint that $\mathcal{F}$ belongs to $[\mathcal{F}_1, \mathcal{F}_2]$ ($\mathcal{F}_1 \leq \mathcal{F} \leq \mathcal{F}_2$), and with the minimization of $\mathcal{F}$ as the objective. This gives us System (2)

Minimize $\mathcal{F}$,
under the constraints

$$\left\{\begin{array}{ll}\text{(2a)} & \forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \;\Rightarrow\; \alpha_{i,j}^{(t)} = 0;\\[2mm] \text{(2b)} & \forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \;\Rightarrow\; \alpha_{i,j}^{(t)} = 0;\\[2mm] \text{(2c)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leq \sup I_t - \inf I_t;\\[2mm] \text{(2d)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1;\\[2mm] \text{(2e)} & \mathcal{F}_1 \leq \mathcal{F} \leq \mathcal{F}_2.\end{array}\right. \qquad (2)$$

6.1.3 *Particular objectives*

The relative ordering of the release dates and deadlines only changes for values of $\mathcal{F}$ where one deadline coincides with a release date or with another deadline. We call such a value of $\mathcal{F}$ a *milestone*.[4] In our problem, there are at most n distinct release dates and as many distinct deadlines. Thus, there are at most $\frac{n(n-1)}{2}$ milestones at which a deadline function coincides with a release date. There are also at most $\frac{n(n-1)}{2}$ milestones at which two deadline functions coincides (two affine functions intersect in at most one point). Let n_{q} be the number of distinct milestones. Then, $1 \leq n_{\text{q}} \leq n^2 - n$. We denote by $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{n_{\text{q}}}$ the milestones ordered by increasing values. To solve our problem we just need to perform a binary search on the set of milestones $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_{n_{\text{q}}}$, each time checking whether System (2) has a solution in the objective interval $[\mathcal{F}_i, \mathcal{F}_{i+1}]$ (except for $i = n_{\text{q}}$ in which case we search for a solution in the range $[\mathcal{F}_{n_{\text{q}}}, +\infty[$). There is a polynomial number of milestones and System (2) can be solved in polynomial time. Therefore, we have:

Theorem 9 (Legrand et al. 2005) *The problem of minimizing the maximum weighted flow time in the divisible load model* $\langle R|r_j;\text{div}|\max w_j F_j\rangle$ *can be solved in polynomial time.*

6.1.4 *A network flow approach for uniform machines*

In Sect. 6.1.1, we presented System (1) to check whether there exists a schedule whose maximum weighted flow is no greater than a given objective. This linear program solves this problem in the unrelated machines case, that is, the most general one. In fact, in the uniform machines framework,

[4] Labetoulle et al. (1984) call such a value a "critical trial value".

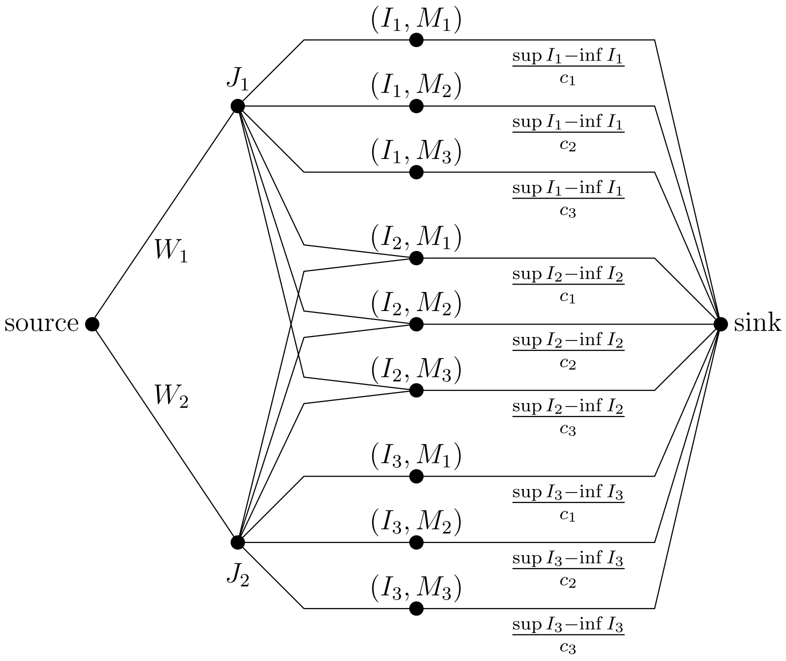

(I_1, M_1)
(I_1, M_2)
(I_1, M_3)
(I_2, M_1)
(I_2, M_2)
(I_2, M_3)
(I_3, M_1)
(I_3, M_2)
(I_3, M_3)
J_1
J_2
W_1
W_2
source
sink
$\frac{\sup I_1 - \inf I_1}{c_1}$
$\frac{\sup I_1 - \inf I_1}{c_2}$
$\frac{\sup I_1 - \inf I_1}{c_3}$
$\frac{\sup I_2 - \inf I_2}{c_1}$
$\frac{\sup I_2 - \inf I_2}{c_2}$
$\frac{\sup I_2 - \inf I_2}{c_3}$
$\frac{\sup I_3 - \inf I_3}{c_1}$
$\frac{\sup I_3 - \inf I_3}{c_2}$
$\frac{\sup I_3 - \inf I_3}{c_3}$

Fig. 3 Graph used to check, on uniform machines and through network flow maximization, whether there exists a schedule of a given maximum weighted flow. This example has two jobs, three machines, and three time intervals defined by the epochal times $r_1 < r_2 < d_1 < d_2$

one can solve this problem using a network flow maximization approach. The graph is built as follows:

Vertices. The graph contains:

- A source
- A sink
- One vertex J_j, for each job J_j, $1 \leq j \leq n$
- One vertex (I_t, M_i) for each ordered pair made of a time interval I_t, $1 \leq t \leq n_{\text{int}}$, and of a machine M_i, $1 \leq i \leq m$

Edges. The graph contains:

- One edge from the source to each node J_j of capacity W_j, the size of the job. This edge represents the amount of work that must be done for the job J_j.
- One edge from each node J_j to each node (I_t, M_i) if, and only if, job J_j can be executed during the time interval I_t (i.e., $r_j \leq \inf I_t$ and $\sup t \leq d_j$). This edge is also of capacity W_j (and is thus not constraining).
- One edge from each node (I_t, M_i) to the sink, of capacity $\frac{\sup I_t - \inf I_t}{c_i}$: this is the amount of work that machine M_i can perform during the time interval I_t.

Figure 3 presents an example of such a graph.

There exists a schedule whose maximum weighted flow is no greater than a given objective $\mathcal{F}$ if the network flow maximization problem for the graph defined above (for the time intervals corresponding to $\mathcal{F}$) has a solution whose flow is equal to $\sum_j W_j$. As previously, one can just check the feasibility of the network flow problem for the *milestones* defined in the previous section. Then, when it is known between which two milestones lies the optimal, the ordering of deadlines is known, and an Earliest Deadline First scheduling leads to an optimal solution. However, this scheme only works in the uniform machines setting as EDF is no longer optimal for uniform machines with restricted availabilities (see the example of Fig. 2). Therefore, we do not know how to use the network flow approach to minimize the max-stretch on uniform machines with restricted availabilities, but this approach can obviously be used in such a framework to check whether a given objective is feasible. Furthermore, this network flow approach cannot be straightforwardly extended to deal with the general case of unrelated machines (even for the problem of uniform machines with restricted availabilities).

6.2 Minimizing the maximum weighted flow with preemption (but no divisibility)

In this section, we focus on the more classical problem with preemption but without the divisible load assumption. We show that combining the linear programming approach of the previous section with the work of Lawler and Labetoulle (1978) leads to a polynomial-time algorithm to solve this problem on unrelated machines. Note that the network flow approach we just recalled enables to minimize the max-stretch with preemption on one machine.

Following the work of Gonzalez and Sahni (1976), Lawler and Labetoulle (1978) present a scheme to build in polynomial-time a preemptive schedule of makespan $\mathcal{C}$ for a set of jobs $J_1, \dots, J_n$ of null release dates ($\forall j, r_j = 0$), under the condition that Linear System (3) has a solution. This system simply states that:

1. all jobs must be fully processed (equation (3a)).
2. the whole processing of a job cannot take a time larger than $\mathcal{C}$ (equation (3b)).
3. the whole utilization time of a machine cannot be longer than a time $\mathcal{C}$ (equation (3c)).

Obviously, these constraints must be satisfied by any preemptive schedule whose makespan is no longer than $\mathcal{C}$. The constructive result obtained by Lawler and Labetoulle shows that such a schedule exists if, and only if, this set of constraints has a solution.

$$\begin{cases} \text{(3a)} \quad \forall j, \quad \sum_{i=1}^{m} \alpha_{i,j} = 1; \\ \text{(3b)} \quad \forall j, \quad \sum_{i=1}^{m} \alpha_{i,j} \cdot p_{i,j} \leq \mathcal{C}; \\ \text{(3c)} \quad \forall i, \quad \sum_{j=1}^{n} \alpha_{i,j} \cdot p_{i,j} \leq \mathcal{C}. \end{cases} \tag{3}$$

Our problem is slightly more general in that we allow arbitrary release dates. Additionally, our objective is to minimize the maximum weighted flow rather than the makespan. Let us consider a maximum weighted flow objective $\mathcal{F}$. As we did in Sect. 6.1.1, we use this objective value to define for each job J_j a deadline $d_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$. As before, the set of release dates and deadlines defines a set of epochal times which, in turn, defines a set of time intervals that we denote by $I_1, \dots, I_{n_{\text{int}}(\mathcal{F})}$. Then, we claim that there exists a preemptive schedule whose maximum weighted flow is no greater than $\mathcal{F}$ if, and only if, Linear System (4) has a solution. Linear System (4) simply states that:

(a) Each job must be processed to completion (equation (4a) which corresponds to equation (3a)).
(b) The processing of a job during the time interval I_t cannot take a time larger than the length of I_t as, in the current framework, a job cannot be simultaneously processed by two different machines (equation (4b) which corresponds to equation (3b)).
(c) The utilization of a machine during a time interval cannot exceed its capacity (equation (4c) which corresponds to equation (3c)).
(d) The processing of a job cannot start before it is released (equation (4d)).
(e) A job must be processed before its deadline (equation (4e)).

$$\begin{cases} \text{(4a)} \quad \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1; \\ \text{(4b)} \quad \forall t, \forall j, \quad \sum_i \alpha_{i,j}^{(t)} . p_{i,j} \leq \sup I_t - \inf I_t; \\ \text{(4c)} \quad \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)} . p_{i,j} \leq \sup I_t - \inf I_t; \\ \text{(4d)} \quad \forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0; \\ \text{(4e)} \quad \forall i, \forall j, \forall t, \quad d_j \leq \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0. \end{cases} \tag{4}$$

Any preemptive schedule whose maximum weighted flow is no greater than $\mathcal{F}$ must obviously satisfy Linear System (4). Conversely, suppose that Linear System (4) has a solution. Then, following Lawler and Labetoulle (1978), we note that the whole system effectively decomposes into a set of linear sub-systems, one for each of the time intervals, and that the sub-system corresponding to interval I_t is exactly equivalent to Linear System (3) where the objective is the length of the time interval (i.e., $\mathcal{C} = \sup I_t - \inf I_t$). Therefore, starting from a solution of Linear System (4) we use the polynomial-time reconstruction scheme of Lawler and Labetoulle to build a preemptive schedule for each of the time intervals I_t. The concatenation of these partial schedules gives us a solution to our problem.

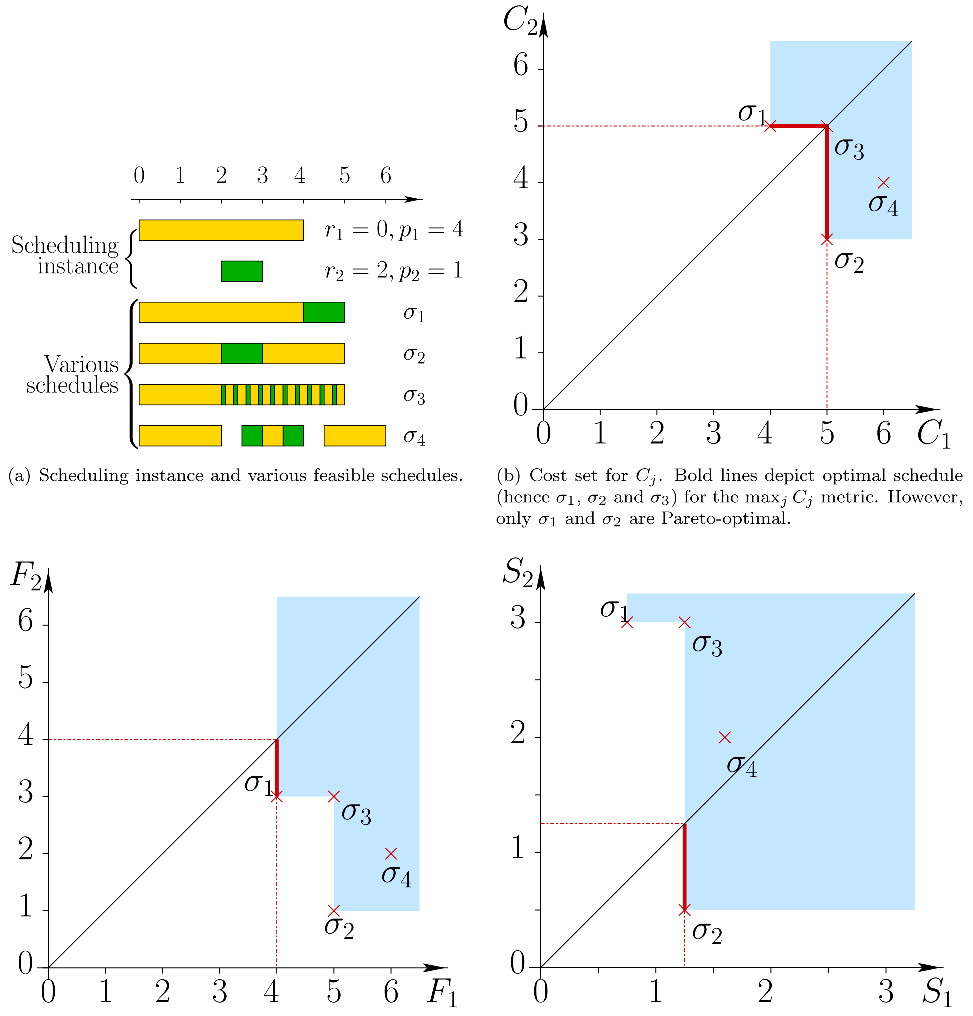
Thus far, we have shown that we are able to check the feasibility of a specific objective value for maximum weighted flow in polynomial time. Moreover, if such an objective is feasible a schedule that achieves this maximum weighted flow can also be built in polynomial time. To finally solve our problem, we recall the methodology presented in Sect. 6.1: Linear System (4) can be used to search for a solution in a range of objective values, defined by consecutive *milestones*, over which the linear system is valid (i.e., the relative order of release dates and deadlines does not change). Similarly, a binary search over the milestones—which are in polynomial number—enables us to find and build an optimal solution in polynomial time. Therefore, we have:

Theorem 10 *The problem of minimizing the maximum weighted flow time in the unrelated machines with preemption model* $\langle R|r_j;\text{pmtn}|\max w_j F_j\rangle$ *can be solved in polynomial time.*

7 Pareto minimization of max-stretch and max flow

In this section we present a few game theory notions and how they translate to our context. This enables us to understand a major flaw of the previous max-based metric in a general framework and how to correctly define a new metric.

Fig. 4 Most optimal solutions to max-based metrics are not Pareto-optimal



(a) Scheduling instance and various feasible schedules.

(b) Cost set for C_j. Bold lines depict optimal schedule (hence σ_1, σ_2 and σ_3) for the $\max_j C_j$ metric. However, only σ_1 and σ_2 are Pareto-optimal.

(c) Cost set for F_j. σ_1 and any schedule such that F_1 and F_2 are smaller than 4 is optimal for $\max_j F_j$. However, only σ_1 is Pareto-optimal for $\max_j F_j$.

(d) Cost set for S_j. σ_2 and any schedule such that S_1 and S_2 are smaller than 1.25 is optimal for $\max_j S_j$. However, only σ_2 is Pareto-optimal for $\max_j S_j$.

Game theory provides a general framework to model situations where many users compete for resources. Each user (in our context, a job) is characterized by a *utility* function u_j. The utility functions represent the satisfaction perceived by the user (typically function of the delay or of the capacity). The goal is to find scheduling strategies such that the utility of *each* user is maximized. In our context it is more relevant to consider cost functions rather than utility functions. Indeed, scheduling problems are typically minimization problems as we try to minimize the completion time, the flow or the stretch of each job. We will therefore assume in the following that the cost γ_j of job J_j is a function of the completion times C. However, as these users may compete for the same resources, it is generally not possible to simultaneously minimize the cost of each user. In a multi-user context, optimality is not defined as simply as in the single-user context, and it is common to use *Pareto-optimality*, defined as follows:

Definition 2 (Pareto-optimality) A vector of completion times $C = (C_1, \ldots, C_n)$ is Pareto-optimal if and only if:

$$\forall \widetilde{C}, \exists i, \quad \gamma_i(\widetilde{C}) < \gamma_i(C) \quad \Longrightarrow \quad \exists j, \quad \gamma_j(C) < \gamma_j(\widetilde{C}).$$

In other words, C is Pareto-optimal if it is impossible to strictly decrease the cost of a player without strictly increasing that of another. Any non-Pareto-optimal schedule can thus be considered as non-efficient as a strictly better usage of resources could be done. Let us consider the cost set $\Gamma \subseteq (\mathbb{R}_+^*)^n$ defined as the set of all feasible cost vectors:

$$\Gamma = \big\{\big(\gamma_1(C), \ldots, \gamma_n(C)\big) \mid \text{there exist a valid schedule with completion times } C\big\}.$$

Figure 4 depicts on each subfigure, for a simple scheduling instance the various cost sets associated to the completion time, flow time, and stretch metrics. The dashed–dotted

Algorithm 2 Heuristic Pareto minimization of max-stretch on one machine

```
1  $FixedStretch \leftarrow \emptyset$
2  $FreeStretch \leftarrow \{J_1, \dots, J_n\}$
3  while $FreeStretch \neq \emptyset$ do
4      Compute the minimum maximum stretch $\mathcal{S}$ of the jobs in $FreeStretch$ taking into account that
       for any job $J_j$ such that $(J_j, \mathcal{S}_j) \in FixedStretch$, $J_j$ has exactly a stretch of $\mathcal{S}_j$.
5      foreach $J_j \in FreeStretch$ do
6          Let $d_j \leftarrow r_j + \mathcal{S} \times p_j$
7      foreach $(J_j, \mathcal{S}_j) \in FixedStretch$ do
8          Let $d_j \leftarrow r_j + \mathcal{S}_j \times p_j$
9      Schedule Earliest Deadline First (EDF) all the jobs (breaking ties randomly); Let $C_j$ be the
       completion time of job $J_j$ under this schedule
10     foreach $J_j \in FreeStretch$ do
11         if $C_j = d_j$ then
12             $FreeStretch \leftarrow FreeStretch \setminus \{J_j\}$
13             $FixedStretch \leftarrow FixedStretch \cup \{(J_j, \mathcal{S})\}$
14 foreach $(J_j, \mathcal{S}_j) \in FixedStretch$ do
15     Let $d_j \leftarrow r_j + \mathcal{S}_j \times p_j$
16 Schedule Earliest Deadline First (EDF) all the jobs
```

line is the optimal isoline for the considered max-based metric ($\max_j C_j$ for Fig. 4b, $\max_j F_j$ for Fig. 4c, and $\max_j S_j$ for Fig. 4d). Any point (the bold lines) belonging to both the isoline and the cost set is thus optimal for the max-based metric. However, we can see that very few of them are Pareto-optimal. This is due to the fact that only the *first* maximum has been minimized. It is well-known in the network community (see, for example, Bertsekas and Gallager 1987; Massoulié and Roberts 2002) that max–min fairness should be *recursively* defined. In our setting, this means that the first maximum should be minimized, then the second should be minimized, and so on. Sum-based metrics obviously do not suffer from this flaw and always produce Pareto-optimal schedules. That is why we propose to consider the new metrics $C_{\max}$ *Pareto*, $F_{\max}$ *Pareto*, and $S_{\max}$ *Pareto*. These scheduling metrics are likely to be much more difficult (but also much more meaningful) than the previous ones as we do not have to only optimize the cost of the more constraining job but to optimize the cost of all jobs at the same time.

7.1 Heuristic Pareto minimization of max-stretch on one machine

Algorithm 2 is an obvious algorithm which recursively tries to minimize the stretch of jobs: First, it minimizes the max-stretch, then the number of jobs whose stretch is equal to the max-stretch, then the maximum stretch of the other jobs, and so on. This algorithm relies on the fact that Earliest Deadline First (EDF) always finds a schedule satisfying all deadlines if one exists (Dertouzos 1974).[5] We show that in some cases Algorithm 2 produces Pareto-optimal schedules for stretch minimization.

Theorem 11 *Algorithm 2 produces a Pareto-optimal schedule for max-stretch minimization on one machine with preemption if at no iteration of the while loop there are two jobs whose deadlines, defined at Steps* 6 *or* 8, *are equal.*

Proofs and details can be found in the research report corresponding to this article (see Legrand et al. 2008). We conjecture that Algorithm 2 always produces a Pareto-optimal schedule for max-stretch minimization on one machine with preemption. This conjecture is based on the facts that (1) the function which associates to a schedule the vector of the stretch of the jobs, sorted in non-decreasing order, is a continuous function; (2) we believe that the set of the instances for which Theorem 11 holds is dense in the space of all instances.

7.2 Heuristic Pareto minimization of max weighted flow on unrelated machines

Here, we target the more general case of the max weighted flow as we will need to look at the special case of max-flow minimization.

[5] On one processor, this property can even be extended to the case of tasks with general dependence relations (Blazewicz 1977).

Algorithm 3 Heuristic Pareto minimization of max weighted flow

```
$FixedWeightedFlow \leftarrow \emptyset$
$FreeWeightedFlow \leftarrow \{J_1, \ldots, J_n\}$
while $FreeWeightedFlow \neq \emptyset$ do
    Compute the minimum max weighted flow $\mathcal{S}$ of the jobs in $FreeWeightedFlow$ taking into
    account that for any job $J_j$ such that $(J_j, \mathcal{S}_j) \in FixedWeightedFlow$, $J_j$ has exactly a weighted flow
    of $\mathcal{S}_j$
    foreach $J_j \in FreeWeightedFlow$ do
        $d_j \leftarrow r_j + \mathcal{S} \times p_j$
    foreach $(J_j, \mathcal{S}_j) \in FixedWeightedFlow$ do
        $d_j \leftarrow r_j + \mathcal{S}_j \times p_j$
    $\mathcal{D} \leftarrow \{d_j \mid S_j \in FreeWeightedFlow\}$
    foreach $d \in \mathcal{D}$ do
        In the set of time intervals defined by the release dates and deadlines (see Section 6.1.1),
        let $I_{t_d}$ be the time interval ending at date $d$: $\sup I_{t_d} = d$
        Solve System (5) (which attempts to complete strictly before $d$ all jobs of deadline $d$)

            $$\begin{array}{l} \textsc{Maximize} \quad \delta \quad , \\ \text{UNDER THE CONSTRAINTS} \\ \begin{cases} \forall i, \forall j, \forall t, \;\; r_j \geqslant \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0; \\ \forall i, \forall j, \forall t, \;\; d_j \leqslant \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0; \\ \forall t, \forall i, \;\; \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leqslant \sup I_t - \inf I_t; \\ \forall j, \;\; \sum_t \sum_i \alpha_{i,j}^{(t)} = 1; \\ \forall i, \;\; \sum_{j|d_j=d} \alpha_{i,j}^{(t_d)}.p_{i,j} \leqslant (\sup I_{t_d} - \inf I_{t_d}) - \delta. \end{cases} \end{array} \qquad (5)$$

        if $\delta = 0$ then
            $S_d \leftarrow \{J_j \in FreeWeightedFlow \mid d_j = d\}$
            Compute a subset $S'_d$ of $S_d$ such that all jobs in $S'_d$ have a max weighted flow of $\mathcal{S}$, and
            such that all the other jobs in $S_d$ can simultaneously have a max weighted flow strictly
            smaller than $\mathcal{S}$.
            foreach $J_j \in S'_d$ do
                $FreeWeightedFlow \leftarrow FreeWeightedFlow \setminus \{J_j\}$
                $FixedWeightedFlow \leftarrow FixedWeightedFlow \cup \{(J_j, \mathcal{S})\}$
foreach $(J_j, \mathcal{S}_j) \in FixedWeightedFlow$ do
    $d_j \leftarrow r_j + \mathcal{S}_j \times p_j$
Build a schedule according to the solution of System (1).
```

Algorithm 3 presents the solution we propose for the general case. The solution for single processor case cannot be straightforwardly extended to the general case as the Earliest Deadline First algorithm is obviously not optimal for non-uniform machines. Once again we (try to) recursively optimize the max weighted flow of the jobs. We compute the best achievable max weighted flow for the jobs whose weighted flow is not yet fixed, and we (try to) minimize the number of jobs whose weighted flow is equal to this maximum. As always the objective max weighted flow gives a deadline per *FreeWeightedFlow* job, i.e., per job whose weighted flow has not yet been fixed. We first minimize the number of distinct deadlines d such that there always is a job whose deadline is d and which is completed at date d. Then we minimize the number of (problematic) jobs, i.e., of jobs which are completed at their deadline.

We can show that Algorithm 3 is correct.

Lemma 3 *Algorithm 3 produces a valid schedule.*

Proofs and details can be found in the research report corresponding to this article (see Legrand et al. 2008).

Step 15 does not explicit how the set “S'_d” of jobs whose completion date equals the deadline should be computed, especially as we would like this set to be as small as possible. In fact, in the general case this problem is NP-complete. The next theorem states the complexity of the Pareto max-flow minimization, and thus of the general case.

Theorem 12 *The Pareto minimization of max-flow on unrelated machines,* $\langle R|\text{div}|F_{\max}\mathit{Pareto}\rangle$, *is NP-complete.*

As we do not have any release dates in the above theorem, we, in fact, prove that $\langle R|\text{div}|C_{\max}\mathit{Pareto}\rangle$ is NP-complete. In fact, we prove an even stronger result, that is, that minimizing the number of jobs whose completion date is equal to the makespan is NP-complete on unrelated machines, and under the divisible model.

This result is proved with a reduction from MINIMUM HITTING SET (Garey and Johnson 1991). The proof can be found in the research report corresponding to this article (see Legrand et al. 2008).

MINIMUM HITTING SET is equivalent to MINIMUM SET COVER (Protasi et al. 1980). Therefore, one of the best polynomial time algorithm to approximate MINIMUM HITTING SET is the greedy algorithm which at each step picks the element which belongs to the largest number of still un-hit subsets. This greedy algorithm has an approximation ratio of $1 + \ln|S|$ (Johnson 1974; Slavík 1996), where $|S|$ is the size of the set.

We do not know what is the complexity of the Pareto minimization of the max-stretch. Seeing how efficient is the greedy heuristic for the minimum hitting set problem, we simply suggest to use it to solve in practice Step 15. Furthermore, one can easily see that when the set S_d at Step 14 is always reduced to a singleton, Algorithm 3 produces an optimal schedule. Therefore:

Theorem 13 *Algorithm* 3 *produces a Pareto-optimal schedule for max weighted flow minimization on unrelated machines under the divisible load model if the set* S_d *at Step* 14 *is always reduced to a singleton.*

We believe that, in practice, the set S_d will always be reduced to a singleton, and thus that Algorithm 3 will always produce optimal schedules. (Note that the case of jobs of same size and same release date is not a problem.)

8 Online max-stretch optimization

In this section, we first improve a lower bound on the competitive ratio of online algorithms for max-stretch minimization established by Bender et al. (1998). Then we present the two competitive algorithms that have previously been proposed in the literature (Bender et al. 1998, 2002). Last we highlight some practical limitations of these algorithms and propose new heuristics that circumvent these limitations.

8.1 Lower bound on the competitiveness of online algorithms

Theorem 14 (Legrand et al. 2006) *There is no* $\frac{1}{2}\Delta^{\sqrt{2}-1}$*-competitive preemptive online algorithm minimizing max-stretch if we restrict to instance with at least three different processing times.*

This result is an improvement from the bound of $\frac{1}{2}\Delta^{\frac{1}{3}}$ established by Bender et al. (1998). In fact, we establish this new bound by doing a more precise analysis of the exact same adversary. In their proof, Bender, Chakrabarti, and Muthukrishnan implicitly assumed that the algorithm knew in advance the ratio Δ of the sizes of the largest and shortest jobs. We will see in the next section that there exist some $O(\sqrt{\Delta})$-competitive algorithms. Therefore, we have roughly bridged half of the gap between the previous lower bound and the best existing algorithms.

Proof We prove this result by contradiction. Therefore, let us assume that there exists a $\frac{1}{2}\Delta^{\sqrt{2}-1}$-competitive preemptive online algorithm $\mathcal{A}$ minimizing max-stretch. An adversary sends the following series of jobs:

Phase 1: Jobs 1 and 2 have both a size of δ and arrive at time 0, i.e., $p_1 = p_2 = \delta$ and $r_1 = r_2 = 0$.

Phase 2: Starting at time $2\delta - k$, and every k time units, a job of size k (with $k < \delta$) arrives. There are x such jobs. In other words, for $1 \le j \le x$, job J_{2+j} arrives at time $r_{2+j} = 2\delta + (j-2)k$ and is of size $p_{2+j} = k$.

A *first come, first served* (FCFS) ordering of all the jobs has a stretch of 2. Algorithm $\mathcal{A}$ is by hypothesis $\frac{1}{2}\Delta^{\sqrt{2}-1}$-competitive and, as a stretch of 2 can be achieved, the date C_1 at which the execution of J_1 ends must satisfy: $\frac{C_1 - r_1}{p_1} \le \frac{1}{2}\Delta^{\sqrt{2}-1} \times 2$ (same constraint on C_2). So far, $\Delta = \frac{\delta}{k}$ (remember that Δ is the ratio of the sizes of the largest and shortest jobs in the system).[6] Therefore, the constraint on C_1 can be rewritten:

$$C_1 \le \frac{1}{2}\Delta^{\sqrt{2}-1} \times 2 \times \delta = \frac{\delta^{\sqrt{2}}}{k^{\sqrt{2}-1}}.$$

The most favorable case for algorithm $\mathcal{A}$ is when it is able to (partially) delay the execution of J_1 and J_2 and to execute

[6] Our bound is tighter than the one established by Bender, Chakrabarti, and Muthukrishnan because we remark that $\Delta = \frac{\delta}{k}$, when they used $\Delta = \frac{\delta}{1} = \delta$, as if they had assumed that the algorithm knew in advance that the ratio of the sizes of the largest and shortest jobs submitted to the system would be δ.

each of the jobs $J_3, \ldots, J_{2+x}$ at its release date. To forbid such a behavior, we choose x, the number of jobs of size k, to be large enough for $\mathcal{A}$ not to be able to delay the completion of J_1 and/or J_2 after the completion of all the jobs of size k. If each of the jobs $J_3, \ldots, J_{2+x}$ is executed at its release date, then $C_{2+x} = 2\delta + (x-1)k$. We define x as follows:

$$x = \left\lfloor 2 + \left(\frac{\delta}{k}\right)^{\sqrt{2}} - \frac{2\delta}{k} \right\rfloor.$$

Then the execution of C_1 and C_2 must be completed by the date $2\delta + (x-1)k$. Otherwise, the algorithm $\mathcal{A}$ fails to achieve its guarantee as the adversary would then send at time $2\delta + (x-1)k$ a job of size $\frac{k+\delta}{2}$ to be exactly under the conditions stated by the theorem. So, algorithm $\mathcal{A}$ must complete the execution of C_1 and C_2 by the date $2\delta + (x-1)k$. Then the adversary sends the following third series of jobs.

Phase 3: Starting at time $2\delta + (x-1)k$, and every time unit, arrives a job of size 1. There are y such jobs. In other words, for $1 \leq j \leq y$, job J_{2+x+j} arrives at time $r_{2+x+j} = 2\delta + (x-1)k + (j-1)$ and is of size $p_{2+x+j} = 1$.

The optimal max-stretch is then less than or equal to $\frac{2\delta+xk+y}{\delta}$ (obtained when delaying the completion of J_1 or J_2 after the completion of all smaller jobs). The max-stretch that algorithm $\mathcal{A}$ can achieve is greater than or equal to $k+1$ when we let $y = \lceil k(k-1) \rceil$. Indeed, the last job completed by algorithm $\mathcal{A}$ is either of size 1 or k and, whatever its size, its stretch is thus the max-stretch of $\mathcal{A}$. Finally, when we pick $k = \delta^{2-\sqrt{2}}$, we obtain the desired contradiction on the competitive ratio of Algorithm $\mathcal{A}$. □

8.2 Competitive online heuristics

We have already seen in Sect. 3.2 that FCFS, the optimal algorithm for the online minimization of max-flow, is only Δ-competitive for the online minimization of max-stretch. This seemingly bad result is obviously partially explained by Theorem 14.

We now recall two existing online algorithms for max-stretch minimization before introducing a new one. Bender et al. (2002) defined, for any job J_j, a pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

$$\widehat{\mathcal{S}}_j(t) = \begin{cases} \frac{t-r_j}{\sqrt{\Delta}} & \text{if } 1 \leq p_j \leq \sqrt{\Delta}, \\ \frac{t-r_j}{\Delta} & \text{if } \sqrt{\Delta} < p_j \leq \Delta. \end{cases}$$

Then, they scheduled the jobs by decreasing pseudo-stretches, potentially preempting running jobs each time a new job arrives in the system. They demonstrated that this method is a $O(\sqrt{\Delta})$-competitive online algorithm.

Bender et al. (1998) had previously described another $O(\sqrt{\Delta})$-competitive online algorithm for max-stretch. This algorithm works as follows: each time a new job arrives, the currently running job is preempted. Then, they compute the optimal (offline) max-stretch $\mathcal{S}^*$ of all jobs having arrived up to the current time. Next, a deadline is computed for each job J_j: $d_j(\mathcal{F}) = r_j + \alpha \times \mathcal{S}^*/p_j$. Finally, a schedule is realized by executing jobs according to their deadlines, using the *Earliest Deadline First* strategy. To optimize their competitive ratio, Bender et al. set their *expansion* factor α to $\sqrt{\Delta}$. For both heuristics, the ratio Δ of the sizes of the largest and shortest jobs submitted to the system is thus assumed to be known in advance.

When they designed their algorithm, Bender et al. did not know how to compute the (offline) optimal maximum stretch. This problem is now overcome. The main remaining problem in this approach, from our point of view, is that such an algorithm tries only to optimize the stretch of the most constraining jobs. This problem is common to all algorithms minimizing a *max* objective. Indeed, such an algorithm may very easily schedule all jobs so that their stretch is equal to the objective, even if most of them could have been scheduled to achieve far lower stretches. This problem is far from being merely theoretical, as we will see in Sect. 9. We will try to circumvent it when designing our own heuristics.

8.3 Practical online heuristics

The basic online heuristic we could derive from our offline algorithm would be along the same line as the algorithm of Bender, Chakrabarti, and Muthukrishnan: each time a new job arrives we would preempt the running job (if any), compute the optimal max-stretch, and schedule the jobs according to the solution of System (2). The solution of System (2) specifies what fraction of each job should be executed on each processor during each time interval. We would implement this solution by breaking arbitrarily the ties that may appear in each time interval.

Our first modification to this scheme is that, rather than computing the "optimal max-stretch", we compute the "best achievable max-stretch considering the decisions already made". In other words, we take into account our knowledge of which jobs were already (partially) executed, and when. The underlying idea being that we cannot change the past. Also, such an optimization will greatly simplify the linear system. This modification is implemented by making trivial modifications to System (2).

Our second modification to the above scheme is more important: we want to optimize more than the max-stretch. The first possibility would be to use in an online framework our offline heuristic for the Pareto minimization of max-stretch. To do so, instead of using a binary search and System (2) to compute the best achievable max-stretch, we use Algorithm 3 where, at Step 4, we compute the best achievable max-stretch rather than the optimal one. This way we define our ONLINE-PARETO heuristics.

Another possible approach would be to specify that each job should be scheduled in a manner that minimizes its own stretch value, while maintaining the overall maximal stretch value obtained. For example, one could theoretically try to minimize the sum-stretch under the condition that the max-stretch be optimal. However, as we have seen, minimizing the sum-stretch is an open problem. So we consider a heuristic approach expressed by System (6).

Minimize

$$\sum_{j=1}^{n} w_j \left(\left(\sum_t \left(\sum_{i=1}^{m} \alpha_{i,j}^{(t)} \right) \frac{\sup I_t(\mathcal{S}^*) + \inf I_t(\mathcal{S}^*)}{2} \right) - r_j \right),$$

under the constraints

$$\begin{cases} \text{(6a)} \quad \forall i, \forall j, \forall t, \quad r_j \geq \sup I_t(\mathcal{S}^*) \Rightarrow \alpha_{i,j}^{(t)} = 0; \\ \text{(6b)} \quad \forall i, \forall j, \forall t, \quad d_j(\mathcal{S}^*) \leq \inf I_t(\mathcal{S}^*) \Rightarrow \alpha_{i,j}^{(t)} = 0; \\ \text{(6c)} \quad \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)} . p_{i,j} \leq \sup I_t(\mathcal{S}^*) - \inf I_t(\mathcal{S}^*); \\ \text{(6d)} \quad \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1. \end{cases} \tag{6}$$

This system ensures that each job is completed no later than the deadline defined by the optimal (offline) max-stretch $\mathcal{S}^*$. Then, under this constraint, this system attempts to minimize an objective that resembles a rational relaxation of the sum-stretch (or more generally of the sum weighted flow) using as an approximation of the completion time, the weighted sum of the average execution times of a job. As we do not know the precise time within an interval when a part of a job will be scheduled, we approximate it by the mean time of the interval. (This heuristic obviously offers no guarantee on the sum-stretch achieved.) This way, we obtain the following online algorithm. Each time a new job arrives:

1. Preempt the running job (if any).
2. Compute the best achievable max-stretch $\mathcal{S}^*$, considering the decisions already made.
3. With the deadlines and intervals defined by the max-stretch $\mathcal{S}^*$, solve System (6).

At this point, we define three variants to produce the schedule. The first, which we call ONLINE, assigns work simply using the values found by the linear program for the α variables:

4. For a given processor P_i, and a given interval $I_t(\mathcal{S}^*)$, all jobs J_j that complete their fraction on that processor during the same interval (i.e., all jobs J_j such that $\sum_{t'>t} \alpha_{i,j}^{(t')} = 0$) are scheduled under the SWRPT policy in that interval. We call these jobs *terminal jobs* (for P_i and $I_t(\mathcal{S}^*)$). The non-terminal jobs scheduled on P_i during interval $I_t(\mathcal{S}^*)$ are only executed in $I_t(\mathcal{S}^*)$ after all terminal jobs have finished.

The second variant we consider, ONLINE-EDF, attempts to make changes to the schedule at the processor level to improve the overall max- and sum-stretch attained:

4. Consider a processor P_i. The fractions $\alpha_{i,j}$ of the jobs that must be partially executed on P_i are processed on P_i under a list scheduling policy based on the following order: the jobs are ordered according to the interval in which their share is completed (according to the solution of the linear program), with ties being broken by the SWRPT policy.

Finally, we propose a third variant, ONLINE-EGDF, that creates a global priority list:

4. The (active) jobs are processed under a list scheduling policy, using the strategy outlined in Sect. 2.3 to deal with restricted availabilities. Here, the jobs are totally ordered by the interval in which their *total work* is completed, with ties being broken by the SWRPT policy.

The validity of these heuristic approaches will be assessed through simulations in the Sect. 9.

9 Simulations

To evaluate the efficacy of various scheduling strategies when optimizing stretch-based metrics, we implemented a simulator using the SimGrid toolkit (Legrand et al. 2003), based on the biological sequence comparison scenario. The application and platform models used in the resulting simulator are derived from our initial observations of the GriPPS system, described in Sect. 2. Our primary goal is to evaluate the proposed heuristics in realistic conditions that include partial replication of target sequence databases across the available computing resources. The remainder of this section outlines the experimental variables we considered and presents results describing the behavior of the heuristics in question under various parametrizations of the platform and application models.

9.1 Simulation settings

The platform and application models that we address in this work are quite flexible, resulting in innumerable variations in the range of potentially interesting combinations. To facilitate our studies, we concretely define certain features of the system that we believe to be useful in describing realistic execution scenarios. We consider in particular six such features.

Platform size Typically, a given biological database such as those considered in this work, would be replicated at various sites, at which comparisons against this database may be performed. Generally, the number of sites in a simulated system provides a basic measure of the aggregate power of the platform. This parameter specifies the exact number of sites in the simulated platform. Without loss of generality, we arbitrarily define each site to contain 10 processors.

Processor power Our model assumes that all the processors at any given site are equivalent, and each processor is assumed to have access to all databases located there. Thus, for each site, a single processor value represents the processing power at that site. We choose processor power values using benchmark results from our previous work.

Number of databases Applications such as GriPPS can accommodate multiple reference databases. Our model allows for any number of distinct databases to exist throughout the system.

Database size Our previous work demonstrated that the processing time needed to service a user request targeting a particular database varies linearly according to the number of sequences found in the database in question. We choose such values from a continuous range of realistic database sizes, with the job size for jobs targeting a particular database scaled accordingly.

Database availability A particular database may be replicated at multiple sites, and a single site may host copies of multiple databases. We account for these two eventualities by associating with each database a probability of existence at each site. The same database availability applies to all databases in the system. We further ensure that each database is available at at least one site, and each site hosts at least one database.

Workload density For a particular database, we define the workload density of a system to be the ratio, on average, of the aggregate job size of user requests against that database to the aggregate computational power available to serve such requests. Workload density expresses a notion of the "load" of the system. This parameter, along with the size of the database, define the frequency of job arrivals in the system.

We define a *simulation configuration* as a set of specific values for each of these six properties. Once defined, concrete *simulation instances* are constructed by realizing random series for any random variables in the system. In particular, two models are created for each instance: a platform model and a workload model. The former is specified first by defining the appropriate number of 10-node sites and assigning corresponding processor power values. Next, a size is assigned to each database, and it is replicated according to the simulation's database availability parameter. Finally, the workload model is realized by first generating a series of jobs for each database, using a Poisson process for job interarrival times, with a mean that is computed to attain the desired workload density. The database-specific workloads are then merged and sorted to obtain the final workload. Jobs may arrive between the time at which the simulation starts and 15 minutes thereafter.

In this simulation study, we use empirical values observed in the GriPPS system logs to define a realistic range of database sizes and to generate appropriate values for processor speeds. The remaining four parameters—platform size, number of distinct databases, database availability, and workload density—are the simulation values that vary in our study. We discuss further the specifics of the experimental design and our simulation results in Sect. 9.3.

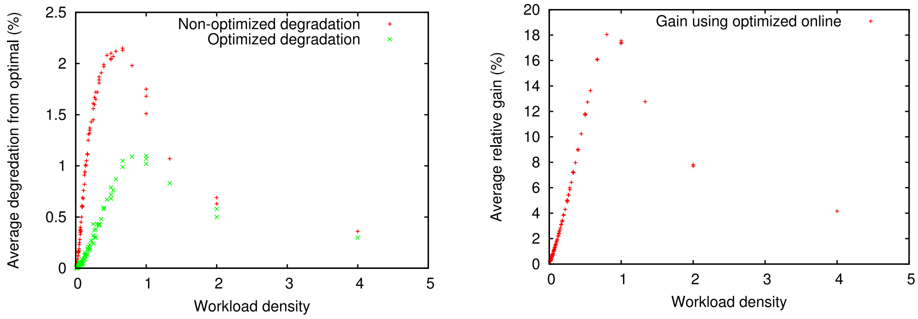
9.2 Optimization of the online heuristic

To motivate the variants of our online heuristic described in Sect. 8, we conduct a series of experiments to evaluate their effect. In particular, we consider a non-optimized version of the online heuristic, which stops after Step 2. We consider job workloads of average density varying between 0.0125 to 4.00, over a range of average job lengths between 15 and 60 seconds. For each job size/workload density combination evaluated, we simulate the execution of 5000 instances, recording the maximum- and sum-stretch of jobs in the workload achieved with both the optimized and non-optimized versions of the online heuristic. The max-stretch of each is then divided by the max-stretch achieved by the optimal algorithm, yielding a degradation factor for both heuristics on that run. Since the optimal sum-stretch is not known, we observe the sum-stretch of the optimized online heuristic relative to the non-optimized version. Figure 5a and b presents the max-stretch and sum-stretch results, respectively. In the first plot, the average max-stretch degradation, compared to the optimal result, for both versions of the heuristic over the 5000 runs of a given configuration is plotted against the workload density of that configuration. The second plot depicts the gain for the sum-stretch metric for the optimized heuristic, relative to the non-optimized version. These results strongly motivate the use of the optimizations encoded by the linear program depicted in System (6).

9.3 Simulation results and analysis

We have implemented in our simulator a number of scheduling heuristics that we plan to compare. First, we have implemented Offline, corresponding to the algorithm described in Sect. 6 that solves the optimal max-stretch

Fig. 5 Comparison of the optimized and non-optimized versions of the online heuristic



(a) Max-stretch degradation from optimal of both versions of the online heuristic.

(b) Sum-stretch gain of the optimized version, relative to the non-optimized version.

problem. Three versions of the online heuristic are also implemented, designated as Online, Online-EDF, and Online-EGDF. Next, we consider the SWRPT, SRPT, and SPT heuristics discussed in Sect. 5. Then, we consider the two online heuristics proposed by Bender et al. that were briefly described in Sect. 8.2. We also include two greedy strategies. First, MCT ("minimum completion time") simply schedules each job as it arrives on the processor that would offer the best job completion time. The FCFS-Div heuristic extends this approach to take advantage of the fact that jobs are divisible, by employing all resources that are able to execute the job (using the strategy laid out in Sect. 2.3). Note that neither MCT nor FCFS-Div makes any changes to work that has already been scheduled. Finally, as a basic reference, we consider a list-scheduling heuristic with random job order denoted RAND. This heuristic works as follows: initially, we randomly build an order on the jobs that may arrive; then RAND list-schedules the jobs while using this list to define priorities, and while using the divisibility property. All the single processor heuristics (SWRPT, SRPT, SPT, and Bender et al.'s) are extended to the multi-processor case using the Algorithm 1 previously described in Sect. 2.3.

As mentioned earlier, two of the six parameters of our model reflect empirical values determined in our previous work with the GriPPS system (Legrand et al. 2005). Processor speeds are chosen randomly from one of the six reference platforms we studied, and we let database sizes vary continuously over a range of 10 megabytes to 1 gigabyte, corresponding roughly to GriPPS database sizes. Thus, our experimental results examine the behaviors of the aforementioned heuristics as we vary our four experimental parameters:

- *platforms* of 3, 10, and 20 clusters (sites) with 10 processors each
- *applications* with 3, 10, and 20 distinct databases
- *database availabilities* of 30%, 60%, and 90% for each database
- *workload density factors* of 0.75, 1.0, 1.25, 1.5, 2.0, and 3.0.

The resulting experimental framework has 162 configurations. For each configuration, 200 platforms and application instances are randomly generated and the simulation results for each of the studied heuristics is recorded. Table 1 presents the aggregate results from these simulations; finer-grained results based on various partitionings of the data may be found in extended version of this document (Legrand et al. 2008).

Above all, we note that the MCT heuristic—effectively the policy in the current GriPPS system—is unquestionably inappropriate for max-stretch optimization: MCT was over 38 times worse, on average, than the best heuristic. Its deficiency might arguably be tolerable on small platforms but, in fact, MCT yielded max-stretch performance over 16 times worse than the best heuristic in all simulation configurations. Even after addressing the primary limitation that the divisibility property is not utilized, the results are still disappointing: FCFS-Div is on average 5.1 times worse in terms of max-stretch than the best approach we found. One of the principal failings of the MCT and FCFS-Div heuristics is that they are non-preemptive. By forcing a small task that arrives in a heavily loaded system to wait, non-preemptive schedulers cause such a task to be inordinately stretched relative to large tasks that are already running.

Experimentally, we find that the first two of the three online heuristics we propose are consistently near-optimal (within 4‰ on average) for max-stretch optimization. The third heuristic, Online-EGDF, actually achieves consistently good sum-stretch (within 3‰ of the best observed sum-stretch), but at the expense of its performance for the max-stretch metric (within 4% of the optimal). This is not entirely surprising as this heuristic ignores a significant portion of the fine-tuned schedule generated by the linear program designed to optimize the max-stretch. Furthermore, our three online heuristics have far better sum-stretch than the Offline-Pareto (which is on average almost 30% away of the best observed sum-stretch). This result validates our heuristic optimization of sum-stretch as expressed by System (6). As forecasted, Offline-Pareto has a significantly better average sum-stretch than Offline.

Table 1 Aggregate statistics over all 162 platform/application configurations

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE	1.0000	0.0000	1.0000	1.4051	0.2784	2.6685
OFFLINE-PARETO	1.0000	0.0000	1.0000	1.2986	0.2605	3.5090
ONLINE	1.0039	0.0145	1.2420	1.0458	0.0439	1.5069
ONLINE-EDF	1.0040	0.0156	1.6886	1.0450	0.0432	1.5016
ONLINE-EGDF	1.0331	0.0622	1.6613	1.0024	0.0052	1.1095
SWRPT	1.0386	0.0729	2.0566	1.0003	0.0014	1.0384
SRPT	1.0596	0.1027	2.1012	1.0048	0.0074	1.1179
SPT	1.0576	0.1032	2.1297	1.0020	0.0048	1.1263
BENDER98*	1.0415	0.0971	2.1521	1.0028	0.0075	1.1393
BENDER02	2.9859	2.7071	23.5446	1.2049	0.3087	6.6820
FCFS-DIV	5.1353	6.6792	65.9073	1.3767	0.7224	15.4213
MCT	38.4276	24.2626	156.3778	51.9606	36.5202	154.1519
RAND	4.6568	6.9107	87.9141	1.2355	0.4827	10.8549

*BENDER98 results are limited to 3-cluster platforms, due to prohibitive overhead costs (discussed in Sect. 9.3)

We also observe that SWRPT, SRPT, and SPT are all quite effective at sum-stretch optimization. Each is on average within 5‰ of the best observed sum-stretch for all configurations. In particular, SWRPT produces a sum-stretch that is on average 0.3‰ within the best observed sum-stretch, and attaining a sum-stretch within 4% of the best sum-stretch in all of the roughly 32,000 instances. However, it should be noted that these heuristics *may* lead to starvation. Jobs may be delayed for an arbitrarily long time, particularly when a long series of small jobs is submitted sequentially (the $(n+1)$th job being released right after the termination of the nth job). Our analysis of the GriPPS application logs has revealed that such situations occur fairly often due to automated processes that submit jobs at regular intervals. By optimizing max-stretch in lieu of sum-stretch, the possibility of starvation is eliminated.

Next, we find that the BENDER98 and BENDER02 heuristics are not practically useful in our scheduling context. The results shown in Table 1 for the BENDER98 heuristic comprise only 3-cluster platforms; simulations on larger platforms were practically infeasible, due to the algorithm's prohibitive overhead costs. Effectively, for an n-task workload, the BENDER98 heuristic solves n optimal max-stretch problems, many of which are computationally equivalent to the full n-task optimal solution. In several cases the desired workload density required thousands of tasks, rendering the BENDER98 algorithm intractable. To roughly compare the overhead costs of the various heuristics, we ran a small series of simulations using only 3-cluster platforms. The results of these tests indicate that the scheduling time for a 15-minute workload was on average under 0.28 s for any of our online heuristics, and 0.54 s for the offline optimal algorithm (with 0.35 s spent in the resolution of the linear program and 0.19 s spent in the online phases of the scheduler); by contrast, the average time spent in the BENDER98 scheduler was 19.76 s. The scheduling overhead of BENDER02 is far less costly (on average 0.23 s of scheduling time in our overhead experiments), but in realistic scenarios for our application domain, the competitive ratios it guarantees are ineffective compared with our online heuristics for max-stretch optimization. Note that the bad performance of BENDER02 is not due to the way we adapt single-machine algorithms to unrelated machines configurations (see Sect. 2.3). Indeed, similar observations can be done when restricting to single-machine configurations (see Table 2).

Finally, we remark that the RAND heuristic is slightly better than the FCFS-DIV for both metrics. Moreover, RAND is only 24% away from the best observed sum-stretch on average. This leads us to think that the sum-stretch may not be a discriminating objective for our problem. Indeed, it looks as if, whatever the policy, any list-scheduling heuristic delivers good performance for this metric.

10 Conclusions

Our initial goal was to minimize the maximum stretch. We have presented a polynomial-time algorithm to solve this problem offline. We have also proposed some heuristics to solve this problem online. Through simulations we have shown that these heuristics are far more efficient than the pre-existing guaranteed heuristics, and do not have the risk of job starvation present in classical simple scheduling heuristics like *shortest remaining processing time*. Along the way we have established some NP-completeness and competitiveness results. Table 3 summarizes the main complexity results presented in this document as well as related work. Minimizing $\max w_j F_j$ is polynomial as soon as divisibility or preemption is allowed, whereas $\sum w_j F_j$ is always

Table 2 Aggregate statistics for a single machine for all application configurations

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE	1.0000	0.0000	1.0000	1.0413	0.0593	1.6735
ONLINE	1.0016	0.0149	1.6344	1.0549	0.0893	1.8134
SWRPT	1.1316	0.2071	3.1643	1.0001	0.0009	1.0398
SRPT	1.1242	0.2003	3.0753	1.0139	0.0212	1.2576
SPT	1.1961	0.2667	3.9752	1.0229	0.0296	1.3573
BENDER98	1.1200	0.1766	2.5428	1.0194	0.0279	1.4466
BENDER02	3.5422	2.4870	21.4819	2.9872	1.9599	15.0019
FCFS-DIV	8.7762	9.1900	80.7465	6.8979	7.7409	88.2449
RAND	11.3059	11.1981	125.3726	5.8227	6.3942	68.0009

Table 3 Summary of complexity results

	$\beta = \emptyset$	$\beta = \text{pmtn}$	$\beta = \text{div}$
$\langle 1 \vert r_j;\beta \vert \max w_j F_j\rangle$	NP (Bender et al. 1998)	$\downarrow$	$\downarrow$
$\langle P \vert r_j;\beta \vert \max w_j F_j\rangle$	$\uparrow$	$\downarrow$	$\downarrow$
$\langle Q \vert r_j;\beta \vert \max w_j F_j\rangle$	$\uparrow$	$\downarrow$	$\downarrow$
$\langle R \vert r_j;\beta \vert \max w_j F_j\rangle$	$\uparrow$	P (Lin. Prog. Sect. 6.2)	P (Lin. Prog. Sect. 6.1)
$\langle 1 \vert r_j;\beta \vert \sum F_j\rangle$	NP (Lenstra et al. 1977)	P (SRPT (Baker 1974))	$\downarrow$
$\langle P \vert r_j;\beta \vert \sum F_j\rangle$	$\uparrow$	NP (Numerical-3DM, Baptiste et al. 2007)	$\downarrow$
$\langle Q \vert r_j;\beta \vert \sum F_j\rangle$	$\uparrow$	$\uparrow$	P (SRPT + Sect. 2.3)
$\langle R \vert r_j;\beta \vert \sum F_j\rangle$	$\uparrow$	$\uparrow$	NP (3DM, Sect. 4)
$\langle 1 \vert r_j;\beta \vert \sum S_j\rangle$	NP (Sect. 5.1)	?	?
$\langle P \vert r_j;\beta \vert \sum S_j\rangle$	$\uparrow$	?	?
$\langle Q \vert r_j;\beta \vert \sum S_j\rangle$	$\uparrow$	?	?
$\langle R \vert r_j;\beta \vert \sum S_j\rangle$	$\uparrow$	?	NP (3DM, Sect. 5.1)
$\langle 1 \vert r_j;\beta \vert \sum w_j F_j\rangle$	NP (Lenstra et al. 1977)	NP (Numerical-3DM, Labetoulle et al. 1984)	$\Lsh$
$\langle P \vert r_j;\beta \vert \sum w_j F_j\rangle$	$\uparrow$	$\uparrow$	$\uparrow$
$\langle Q \vert r_j;\beta \vert \sum w_j F_j\rangle$	$\uparrow$	$\uparrow$	$\uparrow$
$\langle R \vert r_j;\beta \vert \sum w_j F_j\rangle$	$\uparrow$	$\uparrow$	$\uparrow$

strongly NP-hard. $\sum F_j$ is easy only on simple settings (one processor with preemption of related processors with divisibility) and is strongly NP-hard in all other settings. The main problem whose complexity is still open is $\langle 1|r_j, \text{pmtn}|\sum S_j\rangle$ even if (as we already have mentioned in Sect. 5.1) Polynomial Time Approximation Scheme (PTAS) have been proposed for this problem. Some other questions remain open, like:

- What is the complexity of $\langle R|\text{div}; r_j|\mathit{Pareto}S_{\max}\rangle$, the Pareto minimization of max-stretch on unrelated machines under the divisible load model?
- Are there some approximation algorithms minimizing the sum-stretch on unrelated machines under the divisible load model?
- Are there some algorithms with a better competitivity factor than 2 for the minimization of sum-stretch on a single processor?
- Processor Sharing is a scheduling policy where time units are divided arbitrarily finely between jobs and where all jobs currently in the system get an equal share of the machine. In 1998, 1998, Bender et al. claim that Processor Sharing has a competitive ratio of $\Omega(n)$ for max-stretch where n is the number of jobs. This is thus very bad compared to the known $O(\sqrt{\Delta})$ competitive algorithms. However, in the instance they use, Δ grows with n (more precisely $\Delta = 2^n$). Therefore, Processor Sharing may not be such a bad algorithm for the max-stretch minimization. It is not hard (at least numerically) to see that the competitive ratio is $\Omega(\sqrt{\Delta})$. The open question is there-

fore: what is the competitive ratio of Processor Sharing for max-stretch?

Beside all the theoretical considerations, we think that the study presented in this article clearly demonstrates the superiority of our algorithms in terms of fairness and efficiency compared to currently implemented scheduling algorithm in the GriPPS application. In particular, we hope that this study has shown the major importance of divisibility and preemption in this framework and that such techniques will soon be used in practice.

Acknowledgements We would like to thank Lionel Eyraud for the many insightful discussions we had together. We also like to thank Michael Bender who presented us some known algorithms for minimizing max-stretch on one machine with preemption. Last, we would like to thank the reviewers for their thorough reading and insightful comments.

References

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. New York: Wiley.

Baker, K. R., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1983). Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research*, *31*(2), 381–386.

Baptiste, P., Brucker, P., Chrobak, M., Dürr, C., Kravchenko, S. A., & Sourd, F. (2007). The complexity of mean flow time scheduling problems with release times. *Journal of Scheduling*, *10*(2), 139–146.

Bender, M. A. (1998). *New algorithms and metrics for scheduling*. Ph.D. thesis, Harvard University, May 1998.

Bender, M. A., Chakrabarti, S., & Muthukrishnan, S. (1998). Flow and stretch metrics for scheduling continuous job streams. In *Proceedings of the 9th annual ACM-SIAM symposium on discrete algorithms (SODA'98)* (pp. 270–279). Philadelphia: SIAM.

Bender, M. A., Muthukrishnan, S., & Rajaraman, R. (2002). Improved algorithms for stretch scheduling. In *SODA'02: Proceedings of the thirteenth annual ACM-SIAM symposium on discrete algorithms* (pp. 762–771). Philadelphia, PA, USA, 2002. Philadelphia: SIAM.

Bender, M. A., Muthukrishnan, S., & Rajaraman, R. (2004). Approximation algorithms for average stretch scheduling. *Journal of Scheduling*, *7*(3), 195–222.

Bertsekas, D., & Gallager, R. (1987). *Data networks*. Englewood Cliffs: Prentice Hall.

Bharadwaj, V., Ghose, D., Mani, V., & Robertazzi, T. G. (1996). *Scheduling divisible loads in parallel and distributed systems*. Los Alamitos: IEEE Comput. Soc.

Blanchet, C., Combet, C., Geourjon, C., & Deléage, G. (2000). MPSA: integrated system for multiple protein sequence analysis with client/server capabilities. *Bioinformatics*, *16*(3), 286–287.

Blazewicz, J. (1977). Scheduling dependent tasks with different arrival times to meet deadlines. In H. Beilner & E. Gelenbe (Eds.), *Modelling and performance evaluation of computer systems (Proceedings of the international workshop)* (pp. 57–65). Amsterdam: North-Holland.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., & Weglarz, J. (2007). *Handbook on scheduling: from theory to applications*. *International handbooks on information systems*. Berlin: Springer. ISBN: 978-3-540-28046-0.

Braun, R. C., Pedretti, K. T., Casavant, T. L., Scheetz, T. E., Birkett, C. L., & Roberts, C. A. (2001). Parallelization of local BLAST service on workstation clusters. *Future Generation Computer Systems*, *17*(6), 745–754.

Chekuri, C., & Khanna, S. (2002). Approximation schemes for preemptive weighted flow time. In *Proceedings of the 34th annual ACM symposium on theory of computing* (pp. 297–305). New York: Assoc. Comput. Mach.

Darling, A. E., Carey, L., & Feng, W. Ch. (2003). The design, implementation, and evaluation of mpiBLAST. In *Proceedings of ClusterWorld 2003*.

Dertouzos, M. L. (1974). Control robotics: the procedural control of physical processes. In *Proceedings of IFIP congress* (pp. 897–813).

Garey, M. R., & Johnson, D. S. (1991). *Computers and intractability, a guide to the theory of NP-completeness*. New York: Freeman.

Gonzalez, T., & Sahni, S. (1976). Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, *23*(4), 665–679.

GriPPS webpage at http://gripps.ibcp.fr/ (2005).

Johnson, D. S. (1974). Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, *9*, 256–278.

Labetoulle, J., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1984). Preemptive scheduling of uniform machines subject to release dates. In W. R. Pulleyblank (Ed.), *Progress in combinatorial optimization* (pp. 245–261). San Diego: Academic Press.

Lawler, E. L., & Labetoulle, J. (1978). On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the Association for Computing Machinery*, *25*(4), 612–619.

Legrand, A., Marchal, L., & Casanova, H. (2003). Scheduling distributed applications: the SimGrid simulation framework. In *Proceedings of the 3rd IEEE symposium on cluster computing and the grid*.

Legrand, A., Su, A., & Vivien, F. (2004). *Off-line scheduling of divisible requests on an heterogeneous collection of databanks* (Research Report 5386). INRIA, November 2004. Also available as LIP, ENS Lyon, Research Report 2004-51.

Legrand, A., Su, A., & Vivien, F. (2005). Off-line scheduling of divisible requests on an heterogeneous collection of databanks. In *Proceedings of the 14th heterogeneous computing workshop*, Denver, Colorado, USA, April 2005. Los Alamitos: IEEE Comput. Soc.

Legrand, A., Su, A., & Vivien, F. (2006). Minimizing the stretch when scheduling flows of biological requests. In *Symposium on parallelism in algorithms and architectures SPAA'2006*. New York: Assoc. Comput. Mach.

Legrand, A., Su, A., & Vivien, F. (2008). *Minimizing the stretch when scheduling flows of divisible requests* (Research Report RR2008-08). LIP, École Normale Supérieure de Lyon, February 2008. This is a revised version of the LIP Research Report RR2006-19. Also available as INRIA Research Report 6002 http://hal.inria.fr/inria-00108524.

Lenstra, J. K., Rinnooy Kan, A. H. G., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, *1*, 343–362.

Massoulié, L., & Roberts, J. (2002). Bandwidth sharing: objectives and algorithms. *Transactions on Networking*, *10*(3), 320–328.

Megow, N. (2002). *Performance analysis of on-line algorithms in machine scheduling*. Diplomarbeit, Technische Universität Berlin, April 2002.

Miller, P. L., Nadkarni, P. M., & Carriero, N. M. (1991). Parallel computation and FASTA: confronting the problem of parallel database search for a fast sequence comparison algorithm. *Computer Applications in the Biosciences*, *7*(1), 71–78.

Muthukrishnan, S., Rajaraman, R., Shaheen, A., & Gehrke, J. (1999). Online scheduling to minimize average stretch. In *IEEE symposium on foundations of computer science* (pp. 433–442).

Protasi, M. Ausiello, G. D'Atri, A. (1980). Structure preserving reductions among convex optimization problems. *Journal of Computer and System Sciences*, *21*(1), 136–153.

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

Sitters, R. (2005). Complexity of preemptive minsum scheduling on unrelated parallel machines. *Journal of Algorithms*, *57*(1), 37–48.

Slavík, P. (1996). A tight analysis of the greedy algorithm for set cover. In *STOC'96: Proceedings of the twenty-eighth annual ACM symposium on theory of computing* (pp. 435–441). New York: Assoc. Comput. Mach.

Smith, W. E. (1956). Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, *3*, 59–66.