INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Minimizing the stretch when scheduling flows of divisible requests

Arnaud Legrand — Alan Su — Frédéric Vivien

N° 6002

October 2006

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6002--FR+ENG

INRIA
RHÔNE-ALPES

Minimizing the stretch when scheduling flows of divisible requests

Arnaud Legrand , Alan Su , Frédéric Vivien

Thème NUM — Systèmes numériques
Projets GRAAL,MESCAL

Rapport de recherche n° 6002 — October 2006 — 76 pages

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 in 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 uni-processor case, and we show how to extend algorithms that have been proposed in the literature for the uni-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 on-line algorithm, we present new competitiveness results for existing algorithms, and we develop several new on-line heuristics. We also address the Pareto optimization of max-stretch. Then, we extensively study the performance of these algorithms and heuristics in realistic scenarios. Our study shows that all previously proposed guaranteed heuristics for max-stretch for the uni-processor model prove to be inefficient in practice. In contrast, we show our on-line algorithms based on linear programming to be 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.

Key-words: Bioinformatics, heterogeneous computing, scheduling, divisible load, linear programming, stretch

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

Minimisation de l'étirement des tâches lors de l'ordonnancement de flots de requêtes divisibles

Résumé : Dans ce rapport, nous nous intéressons à l'ordonnancement d'applications comparant de manière distribuée des séquences biologiques. Ce problème se situe dans le domaine des tâches divisibles avec coûts de communications négligeables. Jusqu'à présent, très peu de résultats ont été publiés pour ce modèle. Nous discutons et sélectionnons des métriques appropriées pour notre cadre de travail, à savoir le max-stretch et le sum-stretch. Nous expliquons les relations entre notre modèle et le cadre mono-processeur avec préemption, et nous montrons comment étendre au cadre des tâches divisibles sur multi-processeur les algorithmes proposés dans la littérature pour le cas mono-processeur avec préemption. Nous rappelons les résultats connus pour des problématiques proches, nous montrons comment minimiser le max-stretch sur des machines non corrélées (que les tâches soient divisibles ou simplement préemptibles), nous obtenons de nouvelles bornes inférieures de compétitivité pour tout algorithme on-line, nous présentons de nouveaux résultats de compétitivité pour des algorithms de la littérature, et nous proposons de nouvelles heuristiques on-line. Nous nous intéressons également au problème de la minimisation Pareto du max-stretch. Ensuite, nous étudions, de manière extensive, les performances de tous ces algorithmes et de toutes ces heuristiques, et ce dans un cadre réaliste. Notre étude montre que les solutions garanties existantes minimisant le max-stretch sur un processeur sont inefficaces dans notre cadre de travail. Cependant, nous montrons que nos algorithmes on-line basés sur la programmation linéaire ont des performances proches de l'optimal pour le max-stretch. En outre, notre étude suggère clairement les heuristiques qui sont efficaces pour les deux métriques, bien que l'optimisation simultanée pour ces deux métriques soit en théorie impossible dans le cas général.

Mots-clés : Bioinformatique, ordonnancement, tâches divisibles, programmation linéaire, flot pondéré, plates-formes hétérogènes

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 fact, this application is a 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 the genomic sequence comparison scenario, the presence of the required databank on a particular node is the sole factor that constrains task placement decisions. 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., [10, 12, 28]). 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 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 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 Section 2, we review various classical metrics in Section 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 Section 4 we focus on the max-flow and sum-flow metrics. Then in Section 5 we study sum-stretch optimization, in Section 6 offline max-stretch optimization, and in Section 7 Pareto offline optimization of max-stretch. Building on the previous sections, we focus in Section 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, in Section 9 we summarize the known and new results on complexity. Finally, we present in Section 10 an experimental evaluation of the aforementioned heuristics, and we conclude in Section 11.

The main contributions of this work are:

- OFF-LINE SUM-FLOW AND SUM-STRETCH. We show that sum-flow minimization is NP-complete on unrelated machines under the divisible load model ($\langle R|r_j, div|\sum F_j\rangle$ 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 $\langle R|r_j, div|\sum S_j\rangle$ are NP-complete).

- OFF-LINE MAX WEIGHTED FLOW. We present polynomial-time algorithms to solve the minimization of max weighted flow, off-line, on unrelated machines, in the divisible load model and in the preemptive model: $\langle R|r_j; \textit{div}|\max w_j F_j\rangle$ and $\langle R|r_j; \textit{pmtn}|\max w_j F_j\rangle$ are polynomial.

  We also propose heuristics to solve the off-line 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 off-line Pareto minimization of max-flow on unrelated machines is NP-complete.

- ON-LINE SUM-STRETCH AND MAX-STRETCH. We show that FCFS is Δ-competitive for the sum-stretch and max-stretch metrics on one machine, where Δ denotes the ratio of the sizes of the largest and shortest jobs submitted to the system. We also prove that no on-line 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 an approximation algorithm and that *shortest weighted remaining processing time* is at best 2-competitive.

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

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 [9, 17] 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. 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, Su and Vivien [23, 22].

From our modeling perspective, the critical components of this 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 relative to the entire databank. 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 communications.*

   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 communications 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, $p_{i,j}/p_{i',j} = k_{i,i'}$, where $p_{i,j}$ is the time needed to process job J_j on processor i. In essence, $k_{i,i'}$ describes the relative power of processors i and i', regardless of the size or the nature of the job being considered. 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, ..., J_n and m machines (or processors), M_1, ..., 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 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. 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.

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 dependences 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 uni-processor case— we denote by Δ the ratio of the sizes of the largest and shortest jobs submitted to the system: $\Delta = \frac{\max_j W_j}{\min_j W_j}$.

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 (i.e., unrelated machines). 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).

2.3 Relationships with the Uni-Processor Case with Preemption

We first prove that any schedule in the uniform machines model with divisibility has a canonical corresponding schedule in the uni-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, ..., 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 $\widetilde{p} = 1/\sum_i \frac{1}{c_i}$, such that:*

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

Proof. The main idea is that our m heterogeneous processors can be seen as an equivalent processor of power $1/\sum_i \frac{1}{c_i}$. Figure 1 illustrates this idea. More formally, given an instance composed of n jobs J_1, ..., J_n and m machines P_1, ..., P_m such that $p_{i,j} = W_j \cdot c_i$, we define $\widetilde{J_1}$, ..., $\widetilde{J_n}$ with the same release date as the initial jobs and a processing time $\widetilde{p_j} = W_j/(\sum_i \frac{1}{c_i})$. Let us denote by $s^{(t)}$ the time of the t-th preemption and by $\Delta^{(t)}$ the length of time interval before the next preemption. Last, if we define $\alpha_{i,j}^{(t)}$ the fraction of job J_j processed on M_i between the t-th and the $(t+1)$-th preemption (i.e., during the time interval $[s^{(t)}, s^{(t)} + \Delta^{(t)}[$), by construction we have for all P_i: $\sum_j \alpha_{i,j}^{(t)} p_{i,j} \leqslant \Delta^{(t)}$, then $\sum_j \alpha_{i,j}^{(t)} W_j c_i \leqslant \Delta^{(t)}$, hence $\sum_j \alpha_{i,j}^{(t)} W_j \leqslant \frac{\Delta^{(t)}}{c_i}$. Therefore, we have $\sum_i \sum_j \alpha_{i,j}^{(t)} W_j \leqslant \Delta^{(t)} \sum_i \frac{1}{c_i}$ and $\sum_j \left(\sum_i \alpha_{i,j}^{(t)}\right) \frac{W_j}{\sum_i \frac{1}{c_i}} = \sum_j \left(\sum_i \alpha_{i,j}^{(t)}\right) \widetilde{p_j} \leqslant \Delta^{(t)}$.

It is thus possible to process $(\sum_i \alpha_{i,j}^{(t)})$ of job $\widetilde{J_j}$ in time interval $[s^{(t)}, s^{(t+1)}[$, hence defining a valid schedule for our new instance. As preemptions in the new schedule only occur within the ones of the old schedule, no completion time is ever increased. ■

As a consequence, any complexity result for the preemptive uni-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 uni-processor case.

Unfortunately, this line of reasoning is no longer valid when the computational platform exhibits restricted availability, as defined in Section 2. In the uni-processor case, a schedule can be seen as a priority list of the jobs (see the article of Bender, Muthukrishnan, and Rajaraman [7] 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.

The example in Figure 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 Figure 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, so we apply the following simple rule to build a schedule for general platforms from uni-processor heuristics:

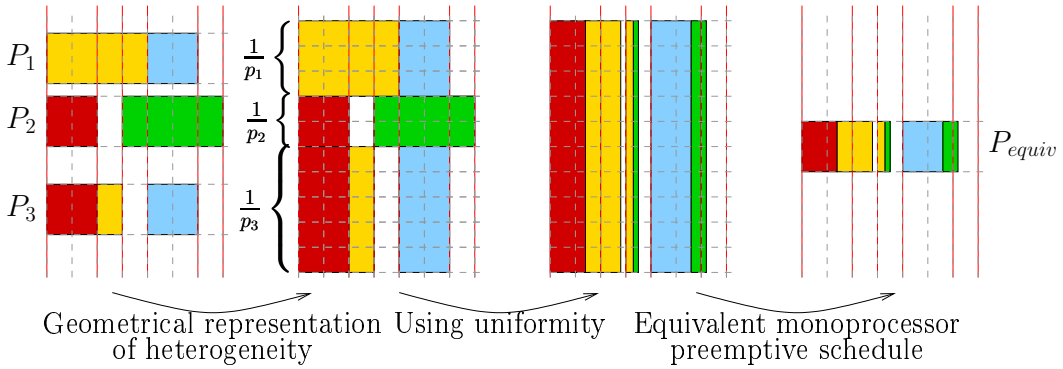


Figure 1: Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem

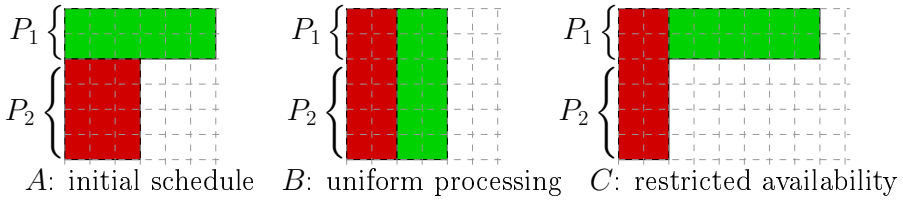


Figure 2: Illustrating the difference between the uniform model and the restricted availability model.

---

1 **while** *some processors are idle* **do**
2 Select the job with the highest priority and distribute its processing on all appropriate processors that are available.

An other 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 relevant to our work (Section 3.1). Then, we show that the optimization of certain objectives are mutually exclusive (Section 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 [5]. 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 [11] 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: $w_j = 1/W_j$ [5]. 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 Sum-Stretch and Max-Stretch Cannot Be Optimized Simultaneously

Finally, we prove that simultaneously optimizing the objectives we have defined earlier (sum-stretch and max-stretch) may be impossible in certain situations. 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.

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. First come, first served *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. [5] for example).*

Proof. We first prove the result for sum-stretch, then for max-stretch, and finally for sum-flow.

1. FCFS is Δ^2 competitive for sum-stretch minimization.

   We first show that FCFS is at worst Δ^2 competitive, then we show that this bound is tight. In this proof, $\mathcal{S}^{\Theta}(\mathcal{I})$ will denote the sum-stretch achieved by the schedule Θ on instance $\mathcal{I}$. $\mathcal{S}^{*}(\mathcal{I})$ will denote the optimal sum-stretch for instance $\mathcal{I}$.

   We show by recurrence on n that for any instance $\mathcal{I} = \{J_1 = (r_1, p_1), ..., J_n = (r_n, p_n)\}$: $\mathcal{S}^{\mathrm{FCFS}}(\mathcal{I}) \leqslant \Delta^2 \mathcal{S}^{*}(\mathcal{I})$. This property obviously holds for $n = 1$. Let us assume that it has been proved for n and prove that it holds true for $n+1$.

   Let us consider $\mathcal{I} = \{J_1 = (r_1, p_1), ..., J_{n+1} = (r_{n+1}, p_{n+1})\}$ an instance with $n+1$ jobs. Without loss of generality, we assume that $\min_j p_j = 1$. We know that, without loss of generality, we may only consider priority list based schedules (see the article of Bender, Muthukrishnan, and Rajaraman [7] for example). Thus let Θ denote the optimal priority list for $\mathcal{I}$. In the following, we denote by A_1 the set of jobs that have a lower priority than J_{n+1} and A_2 the set of jobs that have a higher priority than J_{n+1}. $\rho^{\Theta}(J_k)$ denotes the remaining processing time of J_k at time r_{n+1} under scheduling Θ.

   Thus we have:

$$\mathcal{S}^{\Theta}(J_1, \ldots, J_{n+1}) = \mathcal{S}^{\Theta}(J_1, \ldots, J_n) + \underbrace{\frac{1}{p_{n+1}} \left(p_{n+1} + \sum_{k \in A_1} \rho^{\Theta}(k) \right)}_{\text{The stretch of } J_{n+1}} + \underbrace{\sum_{k \in A_2} \frac{p_{n+1}}{p_k}}_{\text{The cost incurred by } J_{n+1}}$$

We also have:

$$\begin{aligned}
\mathcal{S}^{\text{FCFS}}(J_1,\ldots,J_{n+1}) &= \mathcal{S}^{\text{FCFS}}(J_1,\ldots,J_n) + \underbrace{\frac{1}{p_{n+1}}\left(p_{n+1}+\sum_{k\leqslant n}\rho^{\text{FCFS}}(k)\right)}_{\text{The stretch of } J_{n+1}} \\
&\leqslant \Delta^2\mathcal{S}^*(J_1,\ldots,J_n) + \frac{1}{p_{n+1}}\left(p_{n+1}+\sum_{k\leqslant n}\rho^{\text{FCFS}}(k)\right) \quad \text{(by recurrence hypothesis)} \\
&\leqslant \Delta^2\mathcal{S}^{\Theta}(J_1,\ldots,J_n) + \frac{1}{p_{n+1}}\left(p_{n+1}+\sum_{k\leqslant n}\rho^{\text{FCFS}}(k)\right) \\
&= \Delta^2\mathcal{S}^{\Theta}(J_1,\ldots,J_n) + \frac{1}{p_{n+1}}\left(p_{n+1}+\sum_{k\leqslant n}\rho^{\Theta}(k)\right)
\end{aligned}$$

Indeed, for a priority-based scheduling, at any given time step, the remaining processing time of jobs is independent of the priorities.

$$\mathcal{S}^{\text{FCFS}}(J_1,\ldots,J_{n+1}) \leqslant \Delta^2\mathcal{S}^{\Theta}(J_1,\ldots,J_n) + \frac{1}{p_{n+1}}\left(p_{n+1}+\sum_{k\in A_1}\rho^{\Theta}(k)\right) + \sum_{k\in A_2}\frac{\rho^{\Theta}(k)}{p_{n+1}}$$

As we have $\frac{\rho^{\Theta}(k)}{p_{n+1}} \leqslant \Delta \leqslant \frac{\Delta^2}{\Delta} \leqslant \Delta^2\frac{p_{n+1}}{p_k}$, we get

$$\begin{aligned}
\mathcal{S}^{\text{FCFS}}(J_1,\ldots,J_{n+1}) &\leqslant \Delta^2\mathcal{S}^{\Theta}(J_1,\ldots,J_n) + \Delta^2\left(\frac{1}{p_{n+1}}\left(p_{n+1}+\sum_{k\in A_1}\rho^{\Theta}(k)\right) + \sum_{k\in A_2}\frac{p_{n+1}}{p_k}\right) \\
&\leqslant \Delta^2\mathcal{S}^{\Theta}(J_1,\ldots,J_{n+1}) = \Delta^2\mathcal{S}^*(J_1,\ldots,J_{n+1})
\end{aligned}$$

FCFS is thus Δ^2 competitive for sum-stretch.

We now show that the previous bound is tight by using almost the same construction as in Theorem 1. At date 0 arrives n jobs, J_1, ..., J_n, of size Δ. Then, for any $1 \leqslant j \leqslant n^2$, at time $j-1+\frac{1}{n}$ arrives job J_{n+j} of size 1. A possible schedule would be to process job J_1 during $[0,\frac{1}{n}]$, then each of the jobs J_{n+j}, ..., J_{n+n^2} at its release date, and to wait for the completion of the last of these jobs before completing the jobs J_1, ..., J_n. The sum-stretch would then be $n^2 \times 1 + \frac{n^2+\Delta}{\Delta} + ... + \frac{n^2+n\Delta}{\Delta} = \left(\frac{n(n+1)}{2} + n\frac{n^2}{\Delta}\right) + n^2 = \frac{2n^3+3n^2\Delta+n\Delta}{2\Delta}$. The sum-stretch achieved by FCFS on this instance would be:

$$\frac{\Delta}{\Delta} + ... + \frac{n\Delta}{\Delta} + n^2\left(1+n\Delta-\frac{1}{n}\right) = \frac{n(n+1)}{2} + n^3\Delta + n^2 - n = \frac{2n^3\Delta+3n^2-n}{2}.$$

Therefore, the competitive ratio $\rho(n)$ of FCFS on this instance is:

$$\rho(n) \geqslant \frac{\frac{2n^3\Delta+3n^2-n}{2}}{\frac{2n^3+3n^2\Delta+n\Delta}{2\Delta}} = \frac{\Delta(2n^2\Delta+3n-1)}{2n^2+3n\Delta+\Delta}.$$

Therefore, $\lim_{n\to+\infty} \rho(n) = \Delta^2$. This is all we need to conclude.

2. FCFS is Δ competitive for max-stretch minimization

   We first show that FCFS is at worst Δ competitive, then we show that this bound is tight.

   Let us consider a problem instance J_1, ..., J_n. We denote by C_j the completion time of job J_j under FCFS. We consider any optimal schedule Θ^* for max-stretch. Under this schedule, let C_j^* be the completion time of job J_j, and $\mathcal{S}_j^*$ its stretch. Then, we consider any job J_l which has a larger stretch under FCFS than under Θ^*. Let t be the last time before C_l at which the processor was idle under FCFS. Then, by definition of FCFS, t is the release date r_i of some job J_i and, during the time interval $[r_i, C_l]$, FCFS as exactly executed the jobs J_i, J_{i+1}, ..., J_{l-1}, J_l. Furthermore, by definition of t, during that time interval FCFS neither let the processor idle nor processed, even partially, other jobs. Therefore, as Θ^* completes J_l strictly earlier than C_l, there is a job J_k, $i \leqslant k \leqslant l-1$ that completes at the earliest at time C_l. Then:

$$\max_j \mathcal{S}_j^* \geqslant \mathcal{S}_k^* = \frac{C_k^* - r_k}{p_k} \geqslant \frac{C_l - r_l}{p_k} = \frac{C_l - r_l}{p_l}\frac{p_l}{p_k} \geqslant \mathcal{S}_l \times \frac{1}{\Delta}$$

   Therefore, for any job J_l which has a stretch larger under FCFS than under the optimal schedule Θ^*, $\mathcal{S}_l \leqslant \Delta \times \max_j \mathcal{S}_j^*$. This inequality obviously holds for the other jobs. Hence the upper bound on the competitiveness of FCFS.

   To show that this bound is tight, we use a pretty simple example with two jobs: J_1 arrives at $r_1 = 0$ and has a size of $p_1 = \Delta$, and J_2 which arrives at $r_2 = \varepsilon$ and has a size of $p_2 = 1$. We will have $0 < \varepsilon \ll 1$. FCFS achieves on this example a max-stretch of $1 + \Delta - \varepsilon$. SRPT achieves a max-stretch of $\frac{\Delta+1}{\Delta}$, as $\Delta \geqslant 1$. Hence a competitive ratio for FCFS which is at best:

$$\frac{1+\Delta-\varepsilon}{\frac{\Delta+1}{\Delta}} = \Delta - \varepsilon\frac{\Delta}{\Delta+1} \geqslant \Delta - \varepsilon.$$

   Hence the desired result.

3. FCFS is Δ competitive for sum-flow minimization. This proof follows the same lines than the proof for sum-stretch minimization.

   We first show that FCFS is at worst Δ competitive, then we show that this bound is tight. In this proof, $\mathcal{F}^{\Theta}(\mathcal{I})$ will denote the sum-flow achieved by the schedule Θ on instance $\mathcal{I}$. $\mathcal{F}^*(\mathcal{I})$ will denote the optimal sum-flow for instance $\mathcal{I}$.

We show by recurrence on n that for any instance $\mathcal{I} = \{J_1 = (r_1, p_1), ..., J_n = (r_n, p_n)\}$: $\mathcal{F}^{\text{FCFS}}(\mathcal{I}) \leqslant \Delta\mathcal{F}^*(\mathcal{I})$. This property obviously holds for $n = 1$. Let us assume that it has been proved for n and prove that it holds true for n +1.

Let us consider $\mathcal{I} = \{J_1 = (r_1, p_1), ..., J_{n+1} = (r_{n+1}, p_{n+1})\}$ an instance with $n + 1$ jobs. Without loss of generality, we assume that $\min_j p_j = 1$. We know that, without loss of generality, we may only consider priority list based schedules (see the article of Bender, Muthukrishnan, and Rajaraman [7] for example). Thus let Θ denote the optimal priority list for $\mathcal{I}$. In the following, we denote by A_1 the set of jobs that have a lower priority than J_{n+1} and A_2 the set of jobs that have a higher priority than J_{n+1}. $\rho^{\Theta}(J_k)$ denotes the remaining processing time of J_k at time r_{n+1} under scheduling Θ.

Thus we have:

$$\mathcal{F}^{\Theta}(J_1, \ldots, J_{n+1}) = \mathcal{F}^{\Theta}(J_1, \ldots, J_n) + \underbrace{p_{n+1} + \sum_{k \in A_1} \rho^{\Theta}(k)}_{\text{The flow of } J_{n+1}} + \underbrace{\sum_{k \in A_2} p_{n+1}}_{\text{The cost incrured by } J_{n+1}}$$

We also have:

$$\begin{aligned}
\mathcal{F}^{\text{FCFS}}(J_1, \ldots, J_{n+1}) &= \mathcal{F}^{\text{FCFS}}(J_1, \ldots, J_n) + \underbrace{p_{n+1} + \sum_{k \leqslant n} \rho^{\text{FCFS}}(k)}_{\text{The flow of } J_{n+1}} \\
&\leqslant \Delta\mathcal{F}^*(J_1, \ldots, J_n) + p_{n+1} + \sum_{k \leqslant n} \rho^{\text{FCFS}}(k) \quad \text{(by recurrence hypothesis)} \\
&\leqslant \Delta\mathcal{F}^{\Theta}(J_1, \ldots, J_n) + p_{n+1} + \sum_{k \leqslant n} \rho^{\text{FCFS}}(k) \\
&= \Delta\mathcal{F}^{\Theta}(J_1, \ldots, J_n) + p_{n+1} + \sum_{k \leqslant n} \rho^{\Theta}(k)
\end{aligned}$$

Indeed, for a priority-based scheduling, at any given time step, the remaining processing time of jobs is independent of the priorities.

$$\mathcal{F}^{\text{FCFS}}(J_1, \ldots, J_{n+1}) \leqslant \Delta\mathcal{F}^{\Theta}(J_1, \ldots, J_n) + p_{n+1} + \sum_{k \in A_1} \rho^{\Theta}(k) + \sum_{k \in A_2} \rho^{\Theta}(k)$$

As we have $\rho^{\Theta}(k) \leqslant \Delta \leqslant \Delta p_{n+1}$, we get

$$\begin{aligned}
\mathcal{F}^{\text{FCFS}}(J_1, \ldots, J_{n+1}) &\leqslant \Delta\mathcal{F}^{\Theta}(J_1, \ldots, J_n) + p_{n+1} + \sum_{k \in A_1} \rho^{\Theta}(k) + \sum_{k \in A_2} \Delta p_{n+1} \\
&\leqslant \Delta\mathcal{F}^{\Theta}(J_1, \ldots, J_n) + \Delta p_{n+1} + \Delta \sum_{k \in A_1} \rho^{\Theta}(k) + \Delta \sum_{k \in A_2} p_{n+1}
\end{aligned}$$

$$\leqslant \Delta\mathcal{F}^{\Theta}(J_1,\ldots,J_{n+1}) = \Delta\mathcal{F}^{*}(J_1,\ldots,J_{n+1})$$

We now show that the previous bound is tight by using almost the same construction as in Theorem 1. At date 0 arrives n jobs, J_1, ..., J_n, of size Δ. Then, for any $1 \leqslant j \leqslant n^2$, at time $j-1+\frac{1}{n}$ arrives job J_{n+j} of size 1. A possible schedule would be to process job J_1 during $[0,\frac{1}{n}]$, then each of the jobs J_{n+j}, ..., J_{n+n^2} at its release date, and to wait for the completion of the last of these jobs before completing the jobs J_1, ..., J_n. The sum-flow would then be $n^2\times 1 + n^2 + \Delta + ... + n^2 + n\Delta = n^3 + n^2 + \frac{n(n+1)}{2}\Delta$. The sum-flow achieved by FCFS on this instance would be:

$$\Delta+...+n\Delta+n^2(1+n\Delta-\frac{1}{n}) = \frac{n(n+1)}{2}\Delta+n^3\Delta+n^2-n = \frac{2n^3\Delta+n^2(2+\Delta)+n(\Delta-2)}{2}.$$

Therefore, the competitive ratio $\rho(n)$ of FCFS on this instance is:

$$\rho(n) \geqslant \frac{\frac{2n^3\Delta+n^2(2+\Delta)+n(\Delta-2)}{2}}{n^3+n^2+\frac{n(n+1)}{2}\Delta} = \frac{2n^3\Delta+n^2(2+\Delta)+n(\Delta-2)}{2n^3+2n^2+n(n+1)\Delta}$$

Therefore, $\lim_{n\to+\infty}\rho(n) = \Delta$. This is all we need to conclude. ■

We now prove Theorem 1.

Proof. We first 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).

At date 0 arrives α jobs, J_1, ..., J_α, of size Δ. Let k be any integer. Then, at any time unit t, $0 \leqslant t \leqslant k-1$, 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, ..., J_α. The sum-stretch would then be $k\times 1 + \frac{k+\Delta}{\Delta} + ... + \frac{k+\alpha\Delta}{\Delta} = \frac{\alpha(\alpha+1)}{2} + k\left(1+\frac{\alpha}{\Delta}\right)$ and the max-stretch would be $\alpha + \frac{k}{\Delta}$. Even if it is not optimal for neither one nor the other criteria, 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, ..., J_α 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, ..., J_α 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 $J_{\alpha+1}$, ..., $J_{\alpha+k_1}$ at their release dates, then to schedule J_1, ..., J_α, and then the remaining $k-k_1$ jobs of size 1. The sum-stretch of the actual schedule can therefore not be smaller than the sum-stretch of this schedule, which is equal to:

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

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:

$$\left(\frac{\alpha(\alpha+1)}{2}+\frac{\alpha k_1}{\Delta}\right)+k_1+(k-k_1)(1+\alpha\Delta)\leqslant\rho(\Delta)\left(\frac{\alpha(\alpha+1)}{2}+k\left(1+\frac{\alpha}{\Delta}\right)\right)$$

$$\Leftrightarrow$$

$$-\alpha\Delta k_1+\frac{\alpha(\alpha+1)}{2}(1-\rho(\Delta))+\frac{\alpha k_1}{\Delta}\leqslant k\left(\rho(\Delta)\left(1+\frac{\alpha}{\Delta}\right)-(1+\alpha\Delta)\right).$$

Once the approximation algorithm has completed the execution of the jobs J_1, ..., J_α 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 $\rho(\Delta)\left(1+\frac{\alpha}{\Delta}\right)-(1+\alpha\Delta)\geqslant 0$. 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, ..., J_α 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 everybody's cost (in our context the weighted flow or the stretch of each job) is made as close to the other ones as possible. 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, coupled 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 this setting.

4 Flow Optimization

On a single processor, the max-flow is optimized by FCFS (see Bender *et al.* [5] for example). Using the remarks of Section 2.3, we can thus easily derive an online optimal algorithm for $\langle Q|r_j;\,\textit{div}|F_{\max}\rangle$. We will see in Section 6 that $\langle R|r_j;\,\textit{div}|\max w_jF_j\rangle$ can be solved in polynomial time using linear programing techniques.

Regarding, sum-flow, it was proved by Baker [1], using exchange arguments, that SRPT (shortest remaining processing time first) is optimal for the $\langle 1|r_j;\,\textit{pmtn}|\sum C_j\rangle$ problem. It is thus also optimal for $\langle 1|r_j;\,\textit{pmtn}|\sum F_j\rangle$ and, using the remarks of Section 2.3, we can easily derive an online optimal algorithm for $\langle Q|r_j;\,\textit{div}|\sum F_j\rangle$. 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 [15]:

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

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

Proof. In this section, we present a reduction from *3-Dimensional Matching* to $\langle R|div; r_j|\sum F_j\rangle$. We use the same idea as Sitters [31] used to prove the strong NP-hardness of $\langle R|pmtn|\sum C_j\rangle$. It should be noted that in the reduction we propose, the machines are uniform machines with restricted availabilities. That is why we use W_j for the amount of workload of job j (in Mflop), c_i for the processing capability of machine i (in s per Mflop) and $\delta_{i,j}$ for the availability of machine i to process job j ($\delta_{i,j} \in \{1, +\infty\}$). Therefore, for all i, j, we have $p_{i,j} = W_j.c_i.\delta_{i,j}$.

Scheduling instance. Given an instance of the 3DM problem, we define one machine U_i for each element u_i of the set U. This set of machines is denoted by U as well, and we proceed in the same way for V and W. We also define an additional machine Z. The processing capability of each machine P_i of $U \cup V \cup W \cup Z$ is defined by:

$$c_i = \begin{cases} 1 & \text{if } P_i \in U \cup V \cup W, \\ K/3 & \text{if } P_i = Z \text{ (the exact value of } K \text{ will be precised later).} \end{cases}$$

We define a set of jobs, $(J_j)_{0 \leqslant j \leqslant n-1}$, corresponding to the set S. For each element $s_j = (u_{\alpha_j}, v_{\beta_j}, w_{\gamma_j})$ of S, we define a corresponding job J_j. The size of job J_j is equal to 3 and the processing time of J_j is small on the machines that correspond to the related triplet and on machine Z, and is infinite on all the other ones:

$$\delta_{i,j}^{(J)} = \begin{cases} 1 & \text{if } P_i \in \{U_{\alpha_j}, V_{\beta_j}, W_{\gamma_j}, Z\} \\ \infty & \text{otherwise} \end{cases}, \qquad W_j^{(J)} = 3, \qquad r_j^{(J)} = 0.$$

Hence, we have

$$p_{i,j}^{(J)} = \begin{cases} 3 & \text{if } P_i \in \{U_{\alpha_j}, V_{\beta_j}, W_{\gamma_j}\} \\ K & \text{if } P_i = Z \\ \infty & \text{otherwise} \end{cases}, \qquad r_j^{(J)} = 0.$$

Beside the set J, we also introduce two more sets of jobs: A and B. The set A contains many small jobs that can only be processed on machine Z and whose purpose is to prevent

any job of J from using machine Z before time 2. Therefore, we have $2M$ jobs $(A_j)_{0\leqslant j\leqslant 2M-1}$ that are defined as follows (the exact value of M will be defined later):

$$\delta_{i,j}^{(A)} = \begin{cases} 1 & \text{if } P_i = Z \\ \infty & \text{otherwise} \end{cases}, \qquad W_j^{(A)} = \frac{3}{KM}, \qquad r_j^{(A)} = \frac{j}{M}.$$

Hence, we have

$$p_{i,j}^{(A)} = \begin{cases} 1/M & \text{if } P_i = Z \\ \infty & \text{otherwise} \end{cases}, \qquad r_j^{(A)} = \frac{j}{M}.$$

The set B contains many small jobs that can only be processed on specific machines of U, V, and W and whose purpose is to prevent jobs J from using machines of U, V, and W after time 1. Each machine $u \in U \cup V \cup W$ will be associated to a set of jobs that can only be executed on u. Therefore, we have $N = 3m(nK+2)L$ jobs $(B_j^u)_{\substack{0\leqslant j\leqslant (nK+2)L-1, \\ u\in U\cup V\cup W}}$ that are defined as follows (the exact value of L will be defined later):

$$\delta_{i,j}^{(B^u)} = \begin{cases} 1 & \text{if } P_i = u \\ \infty & \text{otherwise} \end{cases}, \qquad W_j^{(B^u)} = \frac{1}{L}, \qquad r_j^{(B^u)} = 1 + \frac{j}{L}.$$

Hence, we have

$$p_{i,j}^{(B^u)} = \begin{cases} \frac{1}{L} & \text{if } P_i = u \\ \infty & \text{otherwise} \end{cases}, \qquad r_j^{(B)} = 1 + \frac{j}{L}.$$

We now show that the original instance of 3DM has a solution if and only if there is a divisible schedule of the previous instance with sum-flow smaller than or equal to

$$SF_{\text{opt}} = \underbrace{2M \cdot \frac{1}{M}}_{A\text{-jobs}} + \underbrace{m \cdot 1}_{J\text{-jobs from the partition}} + \underbrace{\sum_{k=1}^{n-m} (2 + k.K)}_{J\text{-jobs not from the partition}} + \underbrace{3m(nK+2)L \cdot \frac{1}{L}}_{B\text{-jobs}}$$
$$= 2 + 2n - m + \frac{(n-m).(n-m+1).K}{2} + 3m(nK+2)$$

Equivalence of both problems. Let us start with the easy part:

$\boxed{\Rightarrow}$ Suppose we have a perfect matching $S' \subset S$. Then, we can process all jobs J_j for $s_j \in S'$ on the corresponding machines between time 0 and 1 (see Figure 3). Meanwhile all A-jobs are processed on machine Z between time 0 and 2. The remaining jobs J_j are processed on Z one after the other from time 2 to time $(n-m)K+2$ and all B-jobs are processed on $U \cup V \cup W$ in parallel, one after the other, on their corresponding machines. It is then easy to check that the sum-flow of this schedule is equal to SF_{opt}:

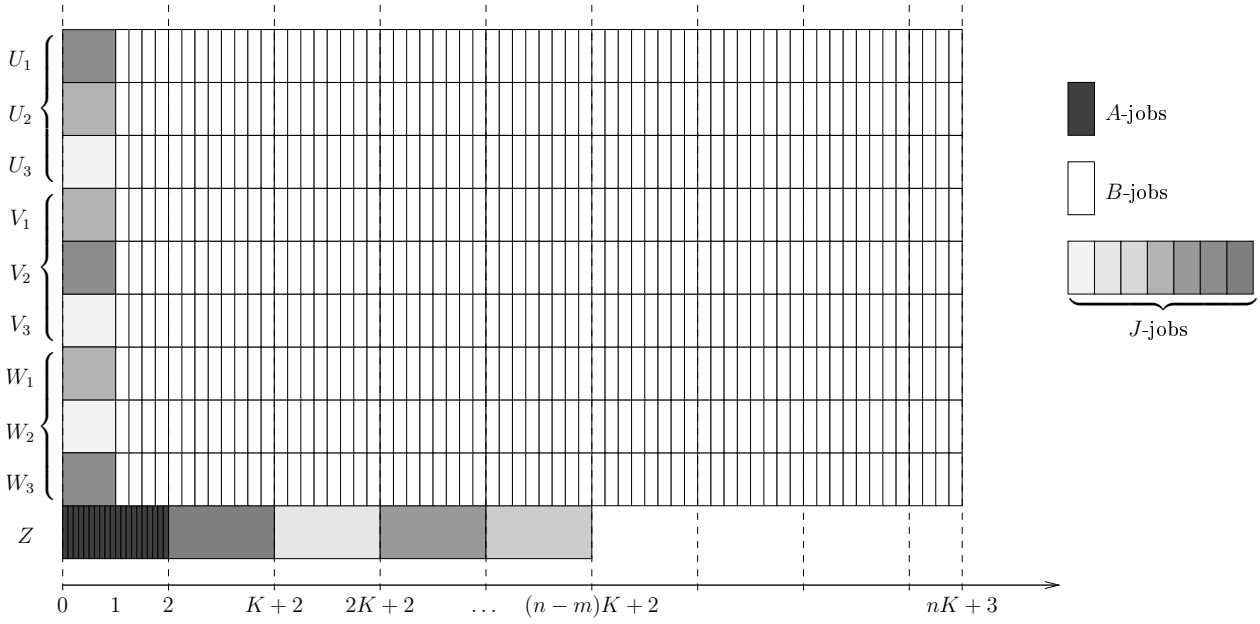


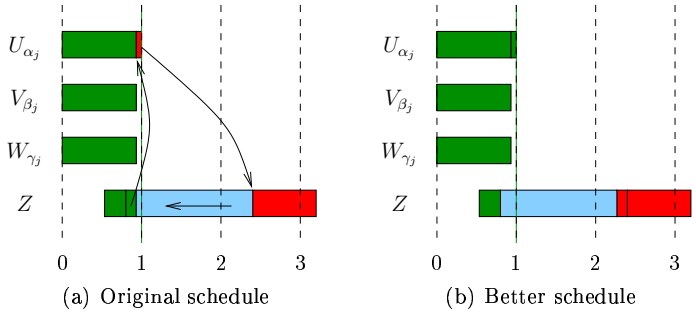
Figure 3: Sketch of the optimal schedule.

$$\begin{cases} F_j^{(J)} = \begin{cases} 1 & \text{if } j \in S' \\ 2+kK & \text{if } j \notin S' \text{ and is the } k\text{-th job to be executed on } Z \end{cases} \\ F_j^{(A)} = \frac{1}{M} \\ F_j^{(B)} = \frac{1}{L} \end{cases}$$

$\boxed{\Leftarrow}$ Let us assume that there is a schedule whose sum-flow is smaller than SF_{opt}. We will first prove that without loss of generality, we can assume that:

1. All B-jobs are processed on machines in $U \cup V \cup W$ during time interval $[1, nK+3]$. As all B-jobs have the same processing time, we can freely assume that on each processor, they are processed in the order of their release dates. Therefore, as these jobs arrive right after each others, if a machine $u \in U \cup V \cup W$ is not used to process B-jobs during a time interval of length $delta$, it will delay all following B_u-jobs by $delta$.

   First we have to see that, even by processing sequentially on Z all tasks that can be processed on Z, any J-job has a completion time smaller than $nK+2$. Therefore if α Mflop of J-jobs are processed on $u \in U \cup V \cup W$ between time 1 and $nK+2$ they necessarily delay at least L B-jobs (the ones that have been



(a) Original schedule (b) Better schedule

Figure 4: No J-job completes before time 1.

released in $[nK+2, nK+3]$)of α. The sum-flow incurred by this processing is therefore larger than $\alpha.L$. By processing this load on Z rather than on u, at most n J-jobs and $2M$ A-jobs would be delayed of $\alpha.K/3$. Therefore as soon as L is larger than $(n+2M)K/3$, we can assume that only B-jobs can be processed on machines $U \cup V \cup W$ during time interval $[1, nK+3]$.

2. J-jobs are processed on machines $U \cup V \cup W$ during time interval $[0,1]$ and on Z during time interval $[0,(n-m)K+2]$; A-jobs are processed by order of their release dates on Z during $[0,(n-m)K+2]$. This is a consequence of the fact that B-jobs are always executed at their release dates. Therefore during the time interval $[1,(n-m)K+2]$, A-jobs and J-jobs are processed on Z as if there was no other machine. Therefore, we can freely assume that they are scheduled by increasing order of their remaining workload, starting at time 1 (SRPT is optimal on a single processor).
3. The completion time of J-jobs is at least 1. Indeed, let us assume that there is a J-job J_j whose completion time equals to $1-\varepsilon < 1$. This means that 3ε units of J_j are processed on Z during $[0, 1-\varepsilon]$ (and therefore we have $\varepsilon \leqslant 1/(K+1) < 1/K$). If U_{α_j} is unused during $[1-\varepsilon, 1]$, then by moving ε units of J_j from Z to U_{α_j}, we increase the completion time of J_j by ε but we also decrease the completion time of at least M jobs by $\varepsilon K/3$. Therefore, we strictly decrease the sum-flow as soon as $K > 3$.
   If U_{α_j} is used during $[1-\varepsilon, 1]$ to process a job $J_{j'}$, then the remaining workload of $J_{j'}$ at time 1 is larger than

$$3 - \left(\left(2+\frac{3}{K}\right)(1-\varepsilon) + \left(3+\frac{3}{K}\right)\varepsilon\right) = 1 - \frac{3}{K} - \varepsilon$$
$$\geqslant 1 - \frac{4}{K} > \frac{3}{KM} \text{ (as soon as } M > 3/(K-4)).$$

   From time 1, only J-jobs and A-jobs are processed on Z. As these jobs are not processed on other machines, we know that these jobs should be processed by

increasing size of the remaining workload. Using above results on the remaining processing workload of J-jobs whose completion time is larger than 1, we can freely assume that all A-jobs are processed first. Therefore (*cf.* Figure 4) by 1) moving ε units of J_j from Z to U_{α_j}, by 2) sliding ε units of A jobs to the left on Z and by 3) moving the units of load of other J-jobs that were processed on U_{α_j} during $[1-\varepsilon, 1]$ next to their corresponding J-jobs on Z, we get a new schedule where 1) the completion of J_j is increased by ε, 2) the completion time of at least M A-jobs is decreased by at least $\varepsilon.\frac{K}{3}$, and 3) the completion time of other J-jobs is unchanged. Hence, this new schedule has a sum flow at least $-\varepsilon + M.\varepsilon.\frac{K}{3}$ smaller than the original one.

4. All A-jobs are processed on machine Z during time interval $[0, 2]$.

   We have just seen that any J-jobs processed on Z has a completion time larger than 1. If α units of J_j are consecutively processed on Z during $[0, 1]$, we can improve the total flow time by moving these units to the last interval where J_j is processed and sliding previous jobs to the left. Therefore we do not increase any completion time and decrease the completion time of at least $\lceil \alpha.MK/3 \rceil$ A-jobs by $\alpha.K/3$, hence a strictly smaller total flow time. We can therefore freely assume that no J-jobs are processed on Z during $[0, 1]$.

   From time 1, we can freely assume that A-jobs and J-jobs are scheduled by increasing order of their remaining workload. If a J-job is processed on Z during $[1, 2]$, it means that its remaining workload is smaller than $3/KM$. Therefore, it has heavily used U_{α_j}, V_{β_j}, and W_{γ_j} and using the same transformation as in Figure 4, we can get a schedule with strictly smaller total flow time. We can therefore assume that only A-jobs are processed on Z during $[0, 2]$.

Therefore by assuming that $K > 3$, $M > 3/(K-4)$, and $L > (n+2M)K/3$, we can freely assume that A-jobs and B-jobs are executed at their release dates on their corresponding processors. Therefore, J-jobs are executed on $U \cup V \cup W$ during $[0, 1]$ and on Z during $[2, (n-m)K+2]$.

Let us now show that our scheduling problem has a solution only if there is a perfect matching for the original 3DM instance. More precisely, we will show that if there is no perfect matching, then the sum-flow is strictly larger than the bound.

Let us denote by x_j the amount of workload of job J_j processed on machines in $U \cup V \cup W$ during time interval $[0, 1]$. We can suppose without loss of generality that:

$$3 \geqslant x_1 \geqslant x_2 \geqslant \cdots \geqslant x_n \geqslant 0.$$

We also have:

$$\sum_{j=1}^{n} x_j \leqslant 3m.$$

The sum-flow of the J-jobs is then equal to:

$$SF(x_1,\ldots,x_n) = \sum_{j=1}^{n}\left(1+\left\lceil\frac{3-x_j}{3}\right\rceil+\sum_{k=1}^{j}K\frac{3-x_k}{3}\right).$$

It is easy to show (using exchange techniques) that under the constraints on the x_j's, $SF(X)$ is *strictly* minimized for $X=(\underbrace{3,\ldots,3}_{m\text{ times}},\underbrace{0,0,\ldots,0}_{n-m\text{ times}})$. Therefore, the sum-flow of a schedule can be equal to SF_{opt} only if $SF(X)=n+(n-m)+K(n-m)(n-m+1)/2$, i.e., only if $X=(3,\ldots,3,0,0,\ldots,)$, which means that there exists a perfect matching for the original 3DM instance. ■

5 Sum-Stretch Optimization

In this section, we give various results regarding sum-stretch optimization. In Section 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.*

Proof. This NP-completeness result is proved by reduction from the version of PARTITION where the two partitions have same size [15]. We first remark that this problem is obviously in NP. Then, let us take an instance $\mathcal{I}_1$ of PARTITION, i.e., a set $\{a_1,a_2,\ldots,a_n\}$ of n integers. Let us denote $B=\frac{1}{2}\sum_{1\leqslant j\leqslant n}a_j$. The question is: is there a subset J of $[1;n]$ such that $\sum_{j\in J}a_j=B$ and such that $|J|=\frac{n}{2}$?

From this instance $\mathcal{I}_1$ of PARTITION, we build the following instance $\mathcal{I}_2$ of $\langle 1|r_j|\sum S_j\rangle$:

- We have n jobs, each of them corresponding to one of the integers in $\mathcal{I}_1$: for $j\in[1;n]$, J_j has size $W_j=B+a_j$ and arrives at time $r_j=0$.
- We have a job J_{n+1} of size $W_{n+1}=\frac{1}{14n^2}$ and which arrives at time $r_{n+1}=\frac{n+2}{2}B$.
- We have $4n^2$ jobs, such that job J_{n+1+j}, $1\leqslant j\leqslant 4n^2$, has size $\frac{1}{14n^2}$ and arrives at time $r_{n+1+j}=(n+2)B+\frac{1}{14n^2}+(j-1)B$.

The question is: is there a schedule of the set of jobs $\{J_j\}_{1\leqslant j\leqslant 4n^2+n+1}$ such that $\sum S_j\leqslant\frac{19}{4}n^2+2n+1$? Note that the size of $\mathcal{I}_2$ is polynomial in the size of $\mathcal{I}_1$.

We first prove that $\mathcal{I}_2$ has a solution if $\mathcal{I}_1$ has one. Therefore, we suppose that there exists a subset J of $[1;n]$ such that $\sum_{j\in J}a_j=B$ and $|J|=\frac{n}{2}$. Then, we build a solution of $\mathcal{I}_2$ has follows:

1. We greedily execute the jobs $\{J_j\}_{1 \leqslant j \leqslant n; j \in J}$ in the time interval $[0; \frac{n+2}{2}B]$. For any such job J_j, we have: $S_j \leqslant \frac{1}{W_j} \cdot \frac{n+2}{2}B \leqslant \frac{1}{B}\frac{n+2}{2}B = \frac{n+2}{2}$.

2. We execute the job J_{n+1} at time $r_{n+1} = \frac{n+2}{2}B$. Its stretch is then 1.

3. We greedily execute the jobs $\{J_j\}_{1 \leqslant j \leqslant n; j \notin J}$ in the time interval $[\frac{n+2}{2}B + \frac{1}{14n^2}; (n+2)B + \frac{1}{14n^2}]$. For any such job J_j we have: $S_j \leqslant \frac{1}{W_j} \cdot \left((n+2)B + \frac{1}{14n^2}\right) \leqslant \frac{1}{B} \cdot ((n+2)B + 1) \leqslant (n+2) + 1$.

4. We execute each of the jobs J_{n+1+j}, $1 \leqslant j \leqslant 4n^2$ at their release date, and each has a stretch of one.

This way we obtain a schedule whose sum-stretch satisfies:

$$\sum S_j \leqslant \frac{n}{2} \cdot \frac{n+2}{2} + 1 + \frac{n}{2} \cdot (n+3) + 4n^2 = \frac{19}{4}n^2 + 2n + 1.$$

Hence we have a solution to problem $\mathcal{I}_2$.

Conversely, let us assume that problem $\mathcal{I}_2$ has a solution. We first show that each of the first n jobs must be completed before the release time of the last job, J_{n+1+4n^2}. We prove this result by contradiction. So, assume that there is a job J_j, with $1 \leqslant j \leqslant n$, which is completed no earlier than time r_{n+1+4n^2}. As the stretch of each of the $n + 4n^2$ other jobs must be at least equal to 1, we derive from our hypothesis:

$$\begin{aligned} \sum S_j &\geqslant n + 4n^2 + \frac{(n+2)B + \frac{1}{14n^2} + (4n^2-1)B}{W_j} \\ &> n + 4n^2 + \frac{(4n^2+n+1)B}{2B} \\ &= 6n^2 + \tfrac{3}{2}n + \tfrac{1}{2} \\ &> \tfrac{19}{4}n^2 + 2n + 1 \end{aligned}$$

Hence our desired contradiction.

We now show that any job, except the first n ones, cannot be delayed by as much as $\frac{1}{2}$ time unit. Indeed, such a delay would induce an increase of stretch of:

$$\frac{\frac{1}{2}}{\frac{1}{14n^2}} = 7n^2 > \frac{19}{4}n^2 + 2n + 1.$$

We now define as J the set of the indices of the jobs which are completed before job J_{n+1}. Obviously, J only contains indices of jobs among the first n ones. As job J_{n+1} must be completed before the date $\frac{n+2}{2}B + \frac{1}{14n^2} + \frac{1}{2}$, $|J| \leqslant \frac{n}{2}$. Furthermore, $|J| = \frac{n}{2}$ if, and only if, $\sum_{j \in J} a_j \leqslant B$. Let J' be the set of the indices of the jobs which are completed after job J_{n+1} is completed and before job J_{n+2} is. From what we have previously shown on the jobs J_{n+1+j}, with $1 \leqslant j \leqslant 4n^2$, we infer that $J \cup J' = \{1, ..., n\}$. Job J_{n+1} is completed at the earliest at time $r_{n+1} + W_{n+1} = \frac{n+2}{2}B + \frac{1}{14n^2}$. Job J_{n+2} is completed no later than at time $r_{n+2} + W_{n+2} + \frac{1}{2} = (n+2)B + \frac{2}{14n^2} + \frac{1}{2}$. Therefore, the set of jobs whose indices are in J' must be executed in a time interval which is shorter than or equal to $\frac{n+2}{2}B + \frac{1}{2} + \frac{1}{14n^2}$.

Therefore $|J'| \leqslant \frac{n}{2}$. Furthermore, $|J'| = \frac{n}{2}$ if, and only if, $\sum_{j \in J'} a_j \leqslant B$. We thus have $|J| = |J'| = \frac{n}{2}$, $\sum_{j \in J} a_j = \sum_{j \in J'} a_j = B$, and J defines a solution to instance $\mathcal{I}_1$. ■

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, Muthukrishnan, and Rajaraman [7] present a Polynomial Time Approximation Scheme (PTAS) for minimizing the sum-stretch with preemption. Chekuri and Khanna [11] 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 $\langle Q|r_j; \mathit{div}|\sum S_j\rangle$ is open (using the remarks of Section 2.3). The minimization of the sum-stretch is however NP-complete on unrelated machines:

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

Proof. In this section, we present a reduction from *3-Dimensional Matching* to $\langle R|\mathit{div}; r_j|\sum S_j\rangle$. We use the same idea as Sitters [31] used to prove the strong NP-hardness of $\langle R|\mathit{pmtn}|\sum C_j\rangle$. It should be noted that in the reduction we propose, the machines are uniform machines with restricted availabilities. That is why we use W_j for the amount of workload of job j (in Mflop), c_i for the processing capability of machine i (in s per Mflop) and $\delta_{i,j}$ for the availability of machine i to process job j ($\delta_{i,j} \in \{1, +\infty\}$). Therefore, for all i, j, we have $p_{i,j} = W_j.c_i.\delta_{i,j}$.

Scheduling instance. Given an instance of the 3DM problem, we define one machine U_i for each element u_i of the set U. This set of machines is denoted by U as well and we proceed in the same way for V and W. We also define an additional machine Z. The processing capability of each machine P_i of $U \cup V \cup W \cup Z$ is defined by:

$$c_i = \begin{cases} 1 & \text{if } P_i \in U \cup V \cup W, \\ K/3 & \text{if } P_i = Z. \end{cases}$$

We define a set of jobs, $(J_j)_{0 \leqslant j \leqslant n-1}$, corresponding to the set S. For each element $s_j = (u_{\alpha_j}, v_{\beta_j}, w_{\gamma_j})$ of S, we define a corresponding job J_j. The size of job J_j is equal to 3 and the processing time of J_j is small on the machines that correspond to the related triplet and on machine Z, and is infinite on all the other ones:

$$\delta_{i,j}^{(J)} = \begin{cases} 1 & \text{if } P_i \in \{U_{\alpha_j}, V_{\beta_j}, W_{\gamma_j}, Z\} \\ \infty & \text{otherwise} \end{cases}, \qquad W_j^{(J)} = 3, \qquad r_j^{(J)} = 0.$$

Hence, we have

$$p_{i,j}^{(J)} = \begin{cases} 3 & \text{if } P_i \in \{U_{\alpha_j}, V_{\beta_j}, W_{\gamma_j}\} \\ K & \text{if } P_i = Z \\ \infty & \text{otherwise} \end{cases}, \qquad r_j^{(J)} = 0, \qquad w_j^{(J)} = \frac{1+1+1+3/K}{3} = \frac{K+1}{K}.$$

Beside the set J, we also introduce two more sets of jobs: A and B. The set A contains many small jobs that can only be processed on machine Z and whose purpose is to prevent any job of J from using machine Z before time 1. Therefore, we have M jobs $(A_j)_{0\leqslant j\leqslant M-1}$ that are defined as follows:

$$\delta_{i,j}^{(A)} = \begin{cases} 1 & \text{if } P_i = Z \\ \infty & \text{otherwise} \end{cases}, \qquad W_j^{(A)} = \frac{3}{KM}, \qquad r_j^{(A)} = \frac{j}{M}.$$

Hence, we have

$$p_{i,j}^{(A)} = \begin{cases} 1/M & \text{if } P_i = Z \\ \infty & \text{otherwise} \end{cases}, \qquad r_j^{(A)} = \frac{j}{M}, \qquad w_j^{(A)} = M.$$

The set B contains many small jobs that can only be processed on machines U, V, and W and whose purpose is to prevent jobs J from using machines of U, V, and W after time 1. Therefore, we have $(n-m)NK$ jobs $(B_j)_{0\leqslant j\leqslant (n-m)NK-1}$ that are defined as follows:

$$\delta_{i,j}^{(B)} = \begin{cases} \infty & \text{if } P_i = Z \\ 1 & \text{otherwise} \end{cases}, \qquad W_j^{(B)} = \frac{3m}{N}, \qquad r_j^{(B)} = 1 + \frac{j}{N}.$$

Hence, we have

$$p_{i,j}^{(B)} = \begin{cases} \infty & \text{if } P_i = Z \\ 3m/N & \text{otherwise} \end{cases}, \qquad r_j^{(B)} = 1 + \frac{j}{N}, \qquad w_j^{(A)} = N.$$

We now show that the original instance of 3DM has a solution if and only if there is a divisible schedule of the previous instance with sum-stretch smaller than or equal to

$$\begin{aligned} SS_{\text{opt}} = & \underbrace{M}_{A\text{-jobs}} + \underbrace{m\frac{K+1}{K}}_{J\text{-jobs from the partition}} + \underbrace{\sum_{k=1}^{n-m}(1+k.K)\cdot\frac{K+1}{K}}_{J\text{-jobs not from the partition}} + \underbrace{(n-m)NK}_{B\text{-jobs}} \\ & = M + n\frac{K+1}{K} + \frac{(n-m)(n-m+1)(K+1)}{2} + (n-m)NK \end{aligned}$$

Equivalence of both problems. Let us start with the easy part:

$\boxed{\Rightarrow}$ Suppose we have a perfect matching $S' \subset S$. Then, we can process all jobs J_j for $s_j \in S'$ on the corresponding machines between time 0 and 1 (see Figure 5). Meanwhile all A-jobs are processed on machine Z. The remaining jobs J_j are processed on Z one after the other from time 1 to time $(n-m)K+1$ and all B-jobs are processed on $U \cup V \cup W$ in order of their release dates, each one being executed in parallel on all the processors of $U \cup V \cup W$. It is then easy to check that the sum-stretch of this schedule

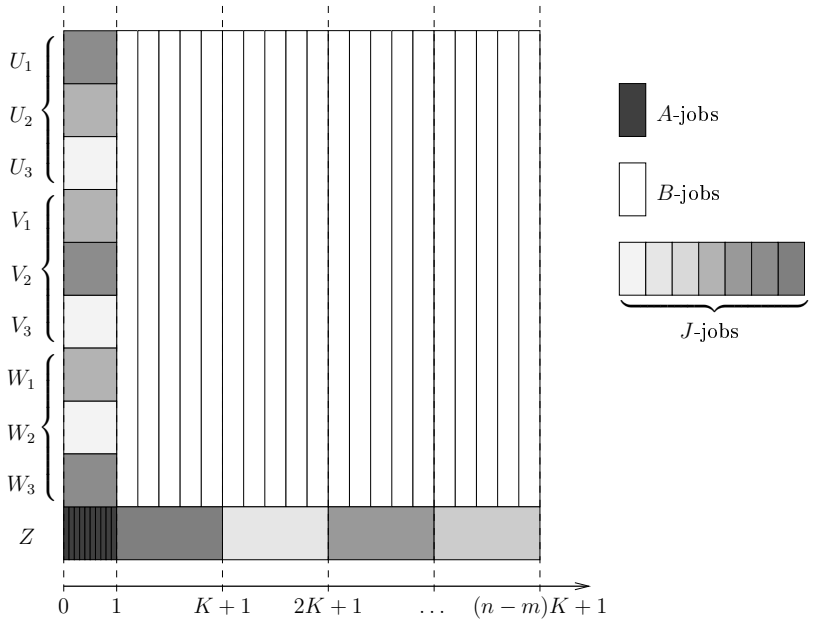


Figure 5: Sketch of the optimal schedule.

is equal to SS_{opt}:

$$\begin{cases} S_j^{(J)} = \begin{cases} \frac{K+1}{K} & \text{if } j \in S' \\ (1+kK)\frac{K+1}{K} & \text{if } j \notin S' \text{ and is the } k\text{-th job to be executed on } Z \end{cases} \\ S_j^{(A)} = 1 \\ S_j^{(B)} = 1 \end{cases}$$

$\boxed{\Leftarrow}$ Let us assume that there is a schedule whose sum-stretch is no greater than SS_{opt}. We will first prove that without loss of generality, we can assume that:

1. All A-jobs are processed on machine Z during time interval $[0,1]$.
   Indeed, let us prove that if we have a schedule where some J-jobs are processed on Z during $[0,1]$, we can transform this schedule in such a way than only A-jobs are processed on Z in $[0,1]$ and the sum-stretch is strictly decreased.
   In Figure 6 situation 1, some A-jobs are delayed by J-jobs during $l > 0$ units of time. Let us consider the stretch difference between situation 1 and 2. In situation 1, at least $\lceil lM \rceil$ A-jobs are delayed of l units of time, which implies that the sum-stretch of these jobs is at least $\lceil lM \rceil \frac{lKM}{3}$ larger than the sum-stretch of

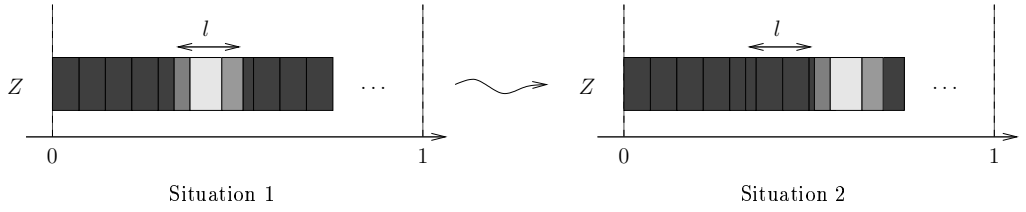


Figure 6: All A-jobs are processed on machine Z during time interval $[0, 1]$.

these jobs in situation 2. In situation 2, the completion-time of J-jobs is at most l units of times larger than in situation 1. The sum-stretch of J-jobs is therefore increased of at most $nl\frac{K+1}{K}$. Situation 1 is thus better than situation 2 only if $\lceil lM \rceil \frac{lKM}{3} < \frac{nl(K+1)}{K}$, i.e., only if $\lceil lM \rceil < \frac{3n(K+1)}{K^2M}$. We will therefore assume in the following that $6n < KM$, hence for situation 1 to be better than situation 2 we must have $\lceil lM \rceil < 1$, which is impossible as $l > 0$. Therefore, delaying A-jobs for executing J-jobs always results in a strict increase of the sum-stretch.

2. All B-jobs are processed on machines in $U \cup V \cup W$ during time interval $[1, (n - m)K + 1]$.

   B-jobs being all equivalent with regards to processing characteristics, they should be executed in the same order as their release dates. They may however be locally preempted by J-jobs. Let us consider B_j, the first B-job that is preempted and is therefore not completely processed during $[r_j^{(B)}, r_{j+1}^{(B)}]$ (see Figure 7 situation 1).

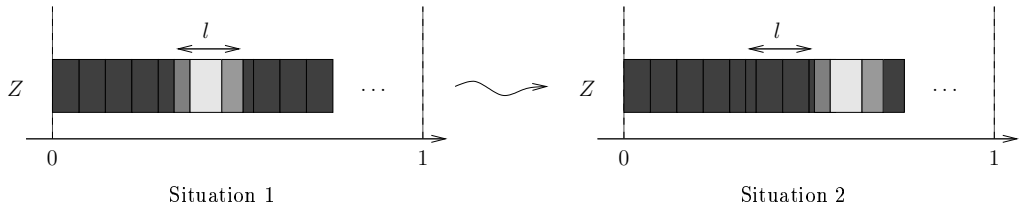


Figure 7: All A-jobs are processed on machine Z during time interval $[0, 1]$.

In Situation 2, all J-jobs that were executed on $U \cup V \cup W$ during $[r_j^{(B)}, r_{j+1}^{(B)}]$ have been transfered on Z. Let us denote by α the amount of J-jobs that has been transfered. The completion time of all J-jobs is therefore at most increased of $\alpha K/3$. The sum-stretch of J-jobs is therefore at most increased of $n\frac{\alpha K}{3}\frac{K+1}{K}$. Likewise, the completion time of B_j is decreased of at least $\frac{\alpha}{3m}$, hence the stretch of B_j is decreased of at least $\frac{\alpha N}{3m}$. Therefore, if we assume in the following that $N > nm(K + 1)$, Situation 2 has always a better sum-stretch than situation 1.

Therefore by assuming that

$$\begin{cases} 6n < KM \\ nm(K+1) < N \end{cases},$$

we can freely assume that A-jobs are executed at their release dates on Z and B-jobs are executed at their release dates on $U \cup V \cup W$. Therefore, J-jobs are executed on $U \cup V \cup W$ during $[0,1]$ and on Z during $[1,(n-m)K+1]$.

Let us now show that our scheduling problem has a solution only if there is a perfect matching for the original 3DM instance. More precisely, we will show that if there is no perfect matching, then the sum-stretch is strictly larger than the bound.

Let us denote by x_j the amount of workload of job J_j processed on machines in $U \cup V \cup W$ during the time interval $[0,1]$. We can suppose without loss of generality that:

$$3 \geqslant x_1 \geqslant x_2 \geqslant \cdots \geqslant x_n \geqslant 0.$$

We also have:

$$\sum_{j=1}^{n} x_j \leqslant 3m$$

As J-jobs all have the same release date, they should be scheduled by increasing size of the remaining workload. Therefore, the sum-stretch of the J-jobs is equal to:

$$SS(x_1,\ldots,x_n) = \sum_{j=1}^{n} \left(1 + \sum_{k=1}^{j} K\frac{3-x_j}{3}\right)\frac{K+1}{K}$$

It is easy to show (using exchange techniques) that under the constraints on the x_j's, $SS(X)$ is *strictly* minimized for $X = (\underbrace{3,\ldots,3}_{m \text{ times}},\underbrace{0,0,\ldots,0}_{n-m \text{ times}})$. Therefore, the sum-stretch of a schedule can be equal to SS_{opt} only if $SS(X) = \frac{n(K+1)}{K} + \frac{(n-m)(n-m+1)(K+1)}{2}$, i.e., only if $X = (3,\ldots,3,0,0,\ldots,)$, which means that there exists a perfect matching for the original 3DM instance. ∎

5.2 Lower Bound on the Competitiveness of Online Algorithms

Muthukrishnan, Rajaraman, Shaheen, and Gehrke [29] 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. *No online algorithm minimizing the sum-stretch with preemption has a competitive ratio less than or equal to 1.19484.*

Proof. We first present the adversary and the analysis of the different possible behaviors for the online algorithms. Finally we will give the optimal values of the different parameters defining the adversary behavior (α, β, γ, n, k, p, ε_1, ε_2, ε_3, and ε_4). The final numerical resolution will show that some of these parameters are not necessary. We decided to present the proof in all its generality rather than to simplify it.

At time $r_0 = 0$ arrives job J_0 of size $p_0 = \alpha\beta\gamma n$.

At time $r_1 = \alpha\beta\gamma n - \varepsilon_1$ arrives job J_1 of size $p_1 = \beta\gamma n$.

We consider the system at time $\alpha\beta\gamma n + \beta\gamma n - \varepsilon_1 - \varepsilon_2$, and whether the execution of J_0 has been completed at that time.

1. If the execution of J_0 has not yet been completed, we do not send any more jobs. To evaluate what is the best achievable sum-stretch in these conditions, we need to study two cases:

   (a) J_0 is completed before J_1. Then, by hypothesis, J_0 cannot be completed earlier than at time $\alpha\beta\gamma n + \beta\gamma n - \varepsilon_1 - \varepsilon_2$. In any case, J_1 is completed at best at time $\alpha\beta\gamma n + \beta\gamma n$. Therefore, the sum-stretch is greater than or equal to:

   $$\frac{\alpha\beta\gamma n + \beta\gamma n - \varepsilon_1 - \varepsilon_2}{\alpha\beta\gamma n} + \frac{(\alpha\beta\gamma n + \beta\gamma n) - (\alpha\beta\gamma n - \varepsilon_1)}{\beta\gamma n} = 2 + \frac{1}{\alpha} - \frac{\varepsilon_1 + \varepsilon_2}{\alpha\beta\gamma n} + \frac{\varepsilon_1}{\beta\gamma n}.$$

   (b) J_1 is completed before J_0. Then J_1 is completed at the earliest at time $r_1 + p_1 = \alpha\beta\gamma n + \beta\gamma n - \varepsilon_1$. In any case, J_0 is completed at best at time $\alpha\beta\gamma n + \beta\gamma n$. Therefore, the sum-stretch is greater than or equal to:

   $$1 + \frac{(\alpha\beta\gamma n + \beta\gamma n)}{\alpha\beta\gamma n} = 2 + \frac{1}{\alpha}.$$

   Therefore, under our hypotheses, the best achievable sum-stretch is:

   $$2 + \frac{1}{\alpha} + \min\left\{0, -\frac{\varepsilon_1 + \varepsilon_2}{\alpha\beta\gamma n} + \frac{\varepsilon_1}{\beta\gamma n}\right\}.$$

   However, one could have completed first J_0 before starting the processing of J_1, hence reaching a sum-stretch of:

   $$2 + \frac{\varepsilon_1}{\beta\gamma n}.$$

   We suppose that (i.e., we will ensure that the chosen values of the parameters are such that):

   $$2 + \frac{\varepsilon_1}{\beta\gamma n} < 2 + \frac{1}{\alpha} + \min\left\{0, -\frac{\varepsilon_1 + \varepsilon_2}{\alpha\beta\gamma n} + \frac{\varepsilon_1}{\beta\gamma n}\right\}.$$

   Then, the competitive ratio attained is greater than or equal to:

   $$\frac{2 + \frac{1}{\alpha} + \min\left\{0, \frac{\varepsilon_1}{\beta\gamma n} - \frac{\varepsilon_1 + \varepsilon_2}{\alpha\beta\gamma n}\right\}}{2 + \frac{\varepsilon_1}{\beta\gamma n}}.$$

2. We now consider the complementary case: at time $\alpha\beta\gamma n+\beta\gamma n-\varepsilon_1-\varepsilon_2$ the execution of J_0 has been completed. Then, we send another job to the system. As J_0 is the first job to be completed, the most favorable case is that J_0 is fully processed during the time interval $[0;\alpha\beta\gamma n]$. We thus assume in the following that this is the case.

   At time $r_2=\alpha\beta\gamma n+\beta\gamma n-\varepsilon_1-\varepsilon_2$ arrives job J_2 of size $p_2=\gamma n$.

   We consider the system at time $\alpha\beta\gamma n+\beta\gamma n+\gamma n-\varepsilon_1-\varepsilon_2-\varepsilon_3$, and whether the execution of J_1 has been completed at that time.

   (a) If the execution of J_1 has not yet been completed, we do not send any more jobs. To evaluate what is the best achievable sum-stretch in these conditions, we need to study two cases:

      i. J_1 is completed before J_2. Then, by hypothesis, J_1 cannot be completed earlier than $\alpha\beta\gamma n+\beta\gamma n+\gamma n-\varepsilon_1-\varepsilon_2-\varepsilon_3$. In any case, J_2 is completed at best at time $\alpha\beta\gamma n+\beta\gamma n+\gamma n$. Therefore, the sum-stretch is greater than or equal to:

$$1+\frac{(\alpha\beta\gamma n+\beta\gamma n+\gamma n-\varepsilon_1-\varepsilon_2-\varepsilon_3)-(\alpha\beta\gamma n-\varepsilon_1)}{\beta\gamma n} + \frac{(\alpha\beta\gamma n+\beta\gamma n+\gamma n)-(\alpha\beta\gamma n+\beta\gamma n-\varepsilon_1-\varepsilon_2)}{\gamma n} = 3+\frac{1}{\beta}-\frac{\varepsilon_2+\varepsilon_3}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}.$$

      ii. J_2 is completed before J_1. Then J_2 is completed at the earliest at time $r_2+p_2=\alpha\beta\gamma n+\beta\gamma n+\gamma n-\varepsilon_1-\varepsilon_2$. In any case, J_1 is completed at best at time $\alpha\beta\gamma n+\beta\gamma n+\gamma n$. Therefore, the sum-stretch is greater than or equal to:

$$1+\frac{(\alpha\beta\gamma n+\beta\gamma n+\gamma n-\varepsilon_1-\varepsilon_2)-(\alpha\beta\gamma n+\beta\gamma n-\varepsilon_1-\varepsilon_2)}{\gamma n} + \frac{(\alpha\beta\gamma n+\beta\gamma n+\gamma n)-(\alpha\beta\gamma n-\varepsilon_1)}{\beta\gamma n} = 3+\frac{1}{\beta}+\frac{\varepsilon_1}{\beta\gamma n}.$$

   Therefore, in that case, the best achievable sum-stretch is

$$3+\frac{1}{\beta}+\min\left\{\frac{\varepsilon_1+\varepsilon_2}{\gamma n}-\frac{\varepsilon_2+\varepsilon_3}{\beta\gamma n},\frac{\varepsilon_1}{\beta\gamma n}\right\}.$$

   However, with only these three jobs, one could have reached a better sum-stretch by first fully execute J_0 and then fully execute J_1. The sum-stretch obtained this

way is equal to:

$$1 + \left(1 + \frac{\varepsilon_1}{\beta\gamma n}\right) + \left(1 + \frac{\varepsilon_1 + \varepsilon_2}{\gamma n}\right) = 3 + \frac{\varepsilon_1}{\beta\gamma n} + \frac{\varepsilon_1 + \varepsilon_2}{\gamma n}.$$

Of course, this is a better solution only if:

$$\frac{\varepsilon_1}{\beta\gamma n} + \frac{\varepsilon_1 + \varepsilon_2}{\gamma n} < \frac{1}{\beta} + \min\left\{\frac{\varepsilon_1 + \varepsilon_2}{\gamma n} - \frac{\varepsilon_2 + \varepsilon_3}{\beta\gamma n}, \frac{\varepsilon_1}{\beta\gamma n}\right\},$$

what we will ensure through the choice of the parameters. In that case, the competitive ratio is greater than or equal to:

$$\frac{3 + \frac{1}{\beta} + \min\left\{\frac{\varepsilon_1+\varepsilon_2}{\gamma n} - \frac{\varepsilon_2+\varepsilon_3}{\beta\gamma n}, \frac{\varepsilon_1}{\beta\gamma n}\right\}}{3 + \frac{\varepsilon_1}{\beta\gamma n} + \frac{\varepsilon_1+\varepsilon_2}{\gamma n}}.$$

(b) We now consider the complementary case: at time $\alpha\beta\gamma n+\beta\gamma n+\gamma n-\varepsilon_1-\varepsilon_2-\varepsilon_3$ the execution of J_1 has been completed. Then, we send another job to the system. As J_0 is the first job to be completed and J_1 the second, the most favorable case is that J_0 is fully processed during the time interval $[0; \alpha\beta\gamma n]$ and J_1 during the time interval $[\alpha\beta\gamma n; \alpha\beta\gamma n + \beta\gamma n]$. We thus assume in the following that this is the case.

At time $r_3 = \alpha\beta\gamma n + \beta\gamma n + \gamma n - \varepsilon_1 - \varepsilon_2 - \varepsilon_3$ **arrives job** J_3 **of size** $p_3 = n$

We consider the system at time $\alpha\beta\gamma n + \beta\gamma n + \gamma_n + n - \varepsilon_1 - \varepsilon_2 - \varepsilon_3 - \varepsilon_4$, and whether the execution of J_2 has been completed at that time.

i. If the execution of J_2 has not yet been completed, we do not send any more jobs. To evaluate what is the best achievable sum-stretch in these conditions, we need to study two cases:

A. J_2 is completed before J_3. Then, by hypothesis, J_2 cannot be completed earlier than $\alpha\beta\gamma n + \beta\gamma n + \gamma n + n - \varepsilon_1 - \varepsilon_2 - \varepsilon_3 - \varepsilon_4$. In any case, J_3 is completed at best at time $\alpha\beta\gamma n + \beta\gamma n + \gamma n + n$. Therefore, the sum-stretch is greater than or equal to:

$$\begin{aligned} &1 + \left(1 + \frac{\varepsilon_1}{\beta\gamma n}\right) \\ &+ \frac{(\alpha\beta\gamma n + \beta\gamma n + \gamma n + n - \varepsilon_1 - \varepsilon_2 - \varepsilon_3 - \varepsilon_4) - (\alpha\beta\gamma n + \beta\gamma n - \varepsilon_1 - \varepsilon_2)}{\gamma n} \\ &+ \frac{(\alpha\beta\gamma n + \beta\gamma n + \gamma n + n) - (\alpha\beta\gamma n + \beta\gamma n + \gamma n - \varepsilon_1 - \varepsilon_2 - \varepsilon_3)}{n} = \\ &4 + \frac{1}{\gamma} + \frac{\varepsilon_1}{\beta\gamma n} - \frac{\varepsilon_3 + \varepsilon_4}{\gamma n} + \frac{\varepsilon_1 + \varepsilon_2 + \varepsilon_3}{n}. \end{aligned}$$

B. J_3 is completed before J_2. Then J_3 is completed at the earliest at time $r_3+p_3 = \alpha\beta\gamma n+\beta\gamma n+\gamma n+n-\varepsilon_1-\varepsilon_2-\varepsilon_3$. In any case, J_2 is completed at best at time $\alpha\beta\gamma n+\beta\gamma n+\gamma n+n$. Therefore, the sum-stretch is greater than or equal to:

$$\begin{aligned}
&1+\left(1+\frac{\varepsilon_1}{\beta\gamma n}\right)\\
&+\frac{(\alpha\beta\gamma n+\beta\gamma n+\gamma n+n-\varepsilon_1-\varepsilon_2-\varepsilon_3)-(\alpha\beta\gamma n+\beta\gamma n+\gamma_n-\varepsilon_1-\varepsilon_2-\varepsilon_3)}{n}\\
&+\frac{(\alpha\beta\gamma n+\beta\gamma n+\gamma n+n)-(\alpha\beta\gamma n+\beta\gamma n-\varepsilon_1-\varepsilon_2)}{\gamma n}\\
&=4+\frac{1}{\gamma}+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}.
\end{aligned}$$

Therefore, in that case, the best achievable sum-stretch is

$$4+\frac{1}{\gamma}+\frac{\varepsilon_1}{\beta\gamma n}+\min\left\{\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}-\frac{\varepsilon_3+\varepsilon_4}{\gamma n},\frac{\varepsilon_1+\varepsilon_2}{\gamma n}\right\}.$$

However, with only these four jobs, one could have reached a better sum-stretch by first fully execute J_0, then fully execute J_1, and then fully execute J_2. The sum-stretch obtained this way is equal to:

$$1+\left(1+\frac{\varepsilon_1}{\beta\gamma n}\right)+\left(1+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}\right)+\left(1+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}\right)=4+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}.$$

Of course, this is a better solution only if:

$$4+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}<4+\frac{1}{\gamma}+\frac{\varepsilon_1}{\beta\gamma n}+\min\left\{\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}-\frac{\varepsilon_3+\varepsilon_4}{\gamma n},\frac{\varepsilon_1+\varepsilon_2}{\gamma n}\right\},$$

what we will ensure through the choice of the parameters. In that case, the competitive ratio is greater than or equal to:

$$\frac{4+\frac{1}{\gamma}+\frac{\varepsilon_1}{\beta\gamma n}+\min\left\{\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}-\frac{\varepsilon_3+\varepsilon_4}{\gamma n},\frac{\varepsilon_1+\varepsilon_2}{\gamma n}\right\}}{4+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}}.$$

ii. We now consider the complementary case: at time $\alpha\beta\gamma n+\beta\gamma n+\gamma n+n-\varepsilon_1-\varepsilon_2-\varepsilon_3-\varepsilon_4$ the execution of J_2 has been completed. Then, we send a series of jobs to the system, as defined below. As J_0 is the first job to be completed, J_1 the second, and J_2 the third, the most favorable case is that J_0 is fully processed during the time interval $[0;\alpha\beta\gamma n]$, J_1 during the time interval $[\alpha\beta\gamma n;\alpha\beta\gamma n+\beta\gamma n]$, and J_2 during the time interval $[\alpha\beta\gamma n+$

$\beta\gamma n; \alpha\beta\gamma n + \beta\gamma n + \gamma n]$. We thus assume in the following that this is the case.

We send to the system a series of k jobs, of same size p, the inter-arrival time of these jobs being equal to p:

At time $r_{3+j} = \alpha\beta\gamma n + \beta\gamma n + \gamma n + n - \varepsilon_1 + (j-1)p$ **arrives job** J_{3+j} **of size** $p_{3+j} = p$**, for** $1 \leqslant j \leqslant k$**.**

Obviously, all the k jobs of size p should be executed in the order of their arrival. The only question to settle is when does the execution of J_3 ends ? We have three cases to consider:

A. The execution of J_3 is completed *before* the execution of any of the jobs J_{3+j}, $1 \leqslant j \leqslant k$. Then the best achievable sum-stretch is equal to:

$$4 + \frac{\varepsilon_1}{\beta\gamma n} + \frac{\varepsilon_1 + \varepsilon_2}{\gamma n} + \frac{\varepsilon_1 + \varepsilon_2 + \varepsilon_3}{n} + k\left(1 + \frac{\varepsilon_1}{p}\right).$$

B. The execution of J_3 is completed *after* the execution of any of the jobs J_{3+j}, $1 \leqslant j \leqslant k$. Then the best achievable sum-stretch is equal to:

$$4 + \frac{\varepsilon_1}{\beta\gamma n} + \frac{\varepsilon_1 + \varepsilon_2}{\gamma n} + \frac{\varepsilon_1 + \varepsilon_2 + \varepsilon_3}{n} + \frac{kp}{n} + k.$$

C. The execution of J_3 is completed between the completion of the jobs J_{3+j} and J_{3+j+1}, for some $j \in [1; k-1]$. Then the best achievable sum-stretch is equal to:

$$4 + \frac{\varepsilon_1}{\beta\gamma n} + \frac{\varepsilon_1 + \varepsilon_2}{\gamma n} + \frac{\varepsilon_1 + \varepsilon_2 + \varepsilon_3}{n} + \frac{jp}{n} + j + (k-j)\left(1 + \frac{\varepsilon_1}{p}\right).$$

This value is obviously a linear combination of the two previous ones, and we do not need to consider that case but just the two extremal ones.

We would like the optimal completion order of the jobs to be J_1, J_2, J_3, J_{3+1}, ..., J_{3+k}, J_0. The sum-stretch for such a schedule is (terms listed in the completion order):

$$\begin{aligned} 1 + \left(1 + \frac{\varepsilon_2}{\gamma n}\right) + \left(1 + \frac{\varepsilon_2 + \varepsilon_3}{n}\right) + k + \frac{\alpha\beta\gamma n + \beta\gamma n + \gamma n + n + kp}{\alpha\beta\gamma n} \\ = k + 4 + \frac{\varepsilon_2}{\gamma n} + \frac{\varepsilon_2 + \varepsilon_3}{n} + \frac{1}{\alpha} + \frac{1}{\alpha\beta} + \frac{1}{\alpha\beta\gamma} + \frac{kp}{\alpha\beta\gamma n}. \end{aligned}$$

Therefore, we want that:

$$\begin{aligned} k + \frac{kp}{\alpha\beta\gamma n} + 4 + \frac{1}{\alpha} + \frac{1}{\alpha\beta} + \frac{1}{\alpha\beta\gamma} + \frac{\varepsilon_2}{\gamma n} + \frac{\varepsilon_2 + \varepsilon_3}{n} \\ < \min \begin{cases} 4 + \frac{\varepsilon_1}{\beta\gamma n} + \frac{\varepsilon_1 + \varepsilon_2}{\gamma n} + \frac{\varepsilon_1 + \varepsilon_2 + \varepsilon_3}{n} + k\left(1 + \frac{\varepsilon_1}{p}\right), \\ 4 + \frac{\varepsilon_1}{\beta\gamma n} + \frac{\varepsilon_1 + \varepsilon_2}{\gamma n} + \frac{\varepsilon_1 + \varepsilon_2 + \varepsilon_3}{n} + \frac{kp}{n} + k. \end{cases} \end{aligned}$$

As we will let k tend to infinity, only the coefficients of k in the different terms matter. Anyway, the competitive ratio in this case is:

$$\frac{\min\left\{4+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}+k\left(1+\frac{\varepsilon_1}{p}\right), 4+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}+\frac{kp}{n}+k\right\}}{k+\frac{kp}{\alpha\beta\gamma n}+4+\frac{1}{\alpha}+\frac{1}{\alpha\beta}+\frac{1}{\alpha\beta\gamma}+\frac{\varepsilon_2}{\gamma n}+\frac{\varepsilon_2+\varepsilon_3}{n}}$$
$$=\frac{k\left(1+\min\left\{\frac{\varepsilon_1}{p},\frac{p}{n}\right\}\right)+4+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}}{k+\frac{kp}{\alpha\beta\gamma n}+4+\frac{1}{\alpha}+\frac{1}{\alpha\beta}+\frac{1}{\alpha\beta\gamma}+\frac{\varepsilon_2}{\gamma n}+\frac{\varepsilon_2+\varepsilon_3}{n}}.$$

Therefore, to find with our construction the largest upper bound on the competitive ratio of online algorithms minimizing the sum-stretch, we need to find values of α, β, γ, n, k, p, ε_1, ε_2, and ε_3, which maximize:

$$\min\left\{\begin{array}{l}
\dfrac{2+\frac{1}{\alpha}+\min\left\{0,\frac{\varepsilon_1}{\beta\gamma n}-\frac{\varepsilon_1+\varepsilon_2}{\alpha\beta\gamma n}\right\}}{2+\frac{\varepsilon_1}{\beta\gamma n}},\\[2ex]
\dfrac{3+\frac{1}{\beta}+\min\left\{\frac{\varepsilon_1+\varepsilon_2}{\gamma n}-\frac{\varepsilon_2+\varepsilon_3}{\beta\gamma n},\frac{\varepsilon_1}{\beta\gamma n}\right\}}{3+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}},\\[2ex]
\dfrac{4+\frac{1}{\gamma}+\frac{\varepsilon_1}{\beta\gamma n}+\min\left\{\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}-\frac{\varepsilon_3+\varepsilon_4}{\gamma n},\frac{\varepsilon_1+\varepsilon_2}{\gamma n}\right\}}{4+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}},\\[2ex]
\dfrac{k\left(1+\min\left\{\frac{\varepsilon_1}{p},\frac{p}{n}\right\}\right)+4+\frac{\varepsilon_1}{\beta\gamma n}+\frac{\varepsilon_1+\varepsilon_2}{\gamma n}+\frac{\varepsilon_1+\varepsilon_2+\varepsilon_3}{n}}{k+\frac{kp}{\alpha\beta\gamma n}+4+\frac{1}{\alpha}+\frac{1}{\alpha\beta}+\frac{1}{\alpha\beta\gamma}+\frac{\varepsilon_2}{\gamma n}+\frac{\varepsilon_2+\varepsilon_3}{n}}.
\end{array}\right.$$

Using Mathematica [26], we found the values $\alpha = 1.93716$, $\beta = 1.29941$, $\gamma = 1$, $n = \frac{1}{\varepsilon_1} \approx 2.69598$, $k = 10^{12}$, $p = 1$, $\varepsilon_1 = 0.370923$, and $\varepsilon_2 = \varepsilon_3 = \varepsilon_4 = 0$ and hence a ratio of 1.19485. ■

5.3 Shortest Remaining Processing Time (SRPT)

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 [29] that SRPT is 2-competitive for sum-stretch.

5.4 Smith's Ratio Rule

Another well studied algorithm is the Smith's ratio rule [33] 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 [30] 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 an approximation algorithm for minimizing the sum-stretch.

Theorem 7. *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 non-null integer. Then, the instance is made of $4\rho + 1$ jobs where job J_k is defined by:

$$\forall k \in [0; 4\rho], \quad r_k = 8\rho k - \frac{k(k+1)}{2} \text{ and } p_k = 8\rho - k.$$

This instance is built such that SPT preempts the execution of job J_k, for $0 \leqslant k \leqslant 4\rho - 1$ one time unit before its completion. Thus the completion of all the jobs is delayed after the completion of $J_{4\rho}$. Then SPT completes the execution of the jobs in the reverse order of their release dates, one every time unit. Therefore, $J_{4\rho}$ has a stretch of 1 and is completed at the date $24\rho^2 + 2\rho$. Then, job J_k, for $0 \leqslant k \leqslant 4\rho - 1$, is completed at time $(24\rho^2 + 2\rho) + (4\rho - k)$ and has a stretch of:

$$\frac{\left((24\rho^2 + 2\rho) + (4\rho - k)\right) - \left(8\rho k - \frac{k(k+1)}{2}\right)}{8\rho - k} \geqslant \frac{24\rho^2 + 6\rho - (8\rho + 1)k}{4\rho}$$

(we roughly bounded the denominator and we just dropped the $\frac{k(k+1)}{2}$ term). Therefore, the sum-stretch reached by SPT on this instance is greater than:

$$1 + \sum_{k=0}^{4\rho-1} \frac{24\rho^2 + 6\rho - (8\rho + 1)k}{4\rho} = 1 + \frac{24\rho^2 + 6\rho}{4\rho}(4\rho) - \frac{8\rho + 1}{4\rho}\frac{(4\rho - 1)4\rho}{2} = 8\rho^2 + 8\rho + \frac{3}{2}.$$

We now need to evaluate the optimal sum-stretch. This optimal is greater than, or equal to, the sum-stretch realized by FIFO scheduling. Under the FIFO scheduling, the job J_k is

completed at time $r_k + p_k + k$ and has thus a stretch of $1 + \frac{k}{p_k} = 1 + \frac{k}{8\rho - k} \leqslant 2$. Therefore, the optimal sum-stretch for this instance is no larger than $2(4\rho + 1)$. Finally, the ratio of the sum-stretch realized by SPT on this instance and of the optimal sum-stretch is greater than or equal to:

$$\frac{8\rho^2 + 8\rho + \frac{3}{2}}{8\rho + 2} > \rho.$$

Note that the largest job, p_0, as a size of 8ρ and the smallest, $p_{4\rho}$, a size of 4ρ. Hence, $\Delta = 2$. ■

5.5 Shortest Weighted Remaining Processing Time (SWRPT)

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, Rajaraman, Shaheen, and Gehrke [29] 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 [27], 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. *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. 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 $\alpha = 1 - \frac{\varepsilon}{3}$, $n = \left\lceil \log_2 \left(\log_2 \frac{3(1+\alpha)}{\varepsilon} \right) \right\rceil$, and $k = \lceil -\log_2(-\log_2 \alpha) \rceil$. Let l be an integer that will be defined later on. 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 $3 \leqslant j \leqslant n$, 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 $1 \leqslant j \leqslant k$, 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 $1 \leqslant j \leqslant l$, is of size $p_{n+k+j} = 1$ and arrives at time $r_{n+k+j} = r_{n+k+j-1} + p_{n+k+j-1}$.

We first study the behavior of SRPT on this instance: this gives us an upper bound on the optimal sum-stretch. Then, we will study the sum-stretch of SWRPT.

***Study of* SRPT.**

- The first date at which SRPT must choose between two jobs is r_1. At r_1 the remaining processing time (RPT) of J_0 is $\rho_{r_1}(J_0) = 2^{2^{n-2}}$, when $\rho_{r_1}(J_1) = 2^{2^{n-1}}$. Therefore, SRPT continues to execute J_0 at date r_1, until $r_1 + 2^{2^{n-2}} < r_2$, at which date the execution of job J_0 is completed.
- We now consider the date r_2.

$$\begin{aligned}\rho_{r_2}(J_1) = 2^{2^{n-1}} - \left(r_2 - \left(r_1 + 2^{2^{n-2}}\right)\right) = \\ 2^{2^{n-1}} - \left(\left(r_1 + 2^{2^{n-1}} - \alpha\right) - \left(r_1 + 2^{2^{n-2}}\right)\right) \\ = 2^{2^{n-1}} - (2^{2^{n-1}} - 2^{2^{n-2}} - \alpha) = 2^{2^{n-2}} + \alpha.\end{aligned}$$

  $\rho_{r_2}(J_2) = 2^{2^{n-2}}$. Therefore, SRPT executes the job J_2 starting at its release date.
- The job J_{2+j}, for $1 \leqslant j \leqslant n + k + l - 2$, is executed at its release date. We can indeed see that at the release date r_{2+j} the only job previously released whose execution was not completed is J_1 whose remaining processing time is $\rho(J_1) = 2^{2^{n-2}} + \alpha$ which is strictly greater than J_{2+j} (the jobs are released in decreasing order of their sizes).
- Once the execution of all the jobs J_{2+j}, for $1 \leqslant j \leqslant n + k + l - 2$, is completed, SRPT completes the execution of job J_1 which ends at time t_f equals to the sum of the sizes of all the jobs:

$$t_f = \sum_{0 \leqslant i \leqslant n} 2^{2^i} + \sum_{1 \leqslant i \leqslant k} 2^{2^{-i}} + l.$$

- From what precedes, the stretch realized by SRPT on this example is equal to one for all the jobs, except for job J_1. Therefore, the sum-stretch realized by SRPT on this instance is equal to:

$$n + k + l - 1 + \frac{t_f - \left(2^{2^n} - 2^{2^{n-2}}\right)}{2^{2^{n-1}}}.$$

***Study of* SWRPT.**

- The first date at which SWRPT must choose between two jobs is r_1. At r_1 the weighted remaining processing time (WRPT, denoted by $\omega_t(J)$) of J_0 is $\omega_{r_1}(J_0) = 2^{2^{n-2}} \times 2^{2^n}$, when $\omega_{r_1}(J_1) = 2^{2^{n-1}} \times 2^{2^{n-1}} = 2^{2^n}$. Therefore, SWRPT preempts job J_0 at date r_1 and executes job J_1 instead.
- We now consider the date r_2.
  - $\omega_{r_2}(J_0) = \omega_{r_1}(J_0) = 2^{2^{n-2}} \times 2^{2^n}$.
  - $\omega_{r_2}(J_1) = \left(2^{2^{n-1}} - (r_2 - r_1)\right) \times 2^{2^{n-1}} = \left(2^{2^{n-1}} - \left(2^{2^{n-1}} - \alpha\right)\right) \times 2^{2^{n-1}} = \alpha \times 2^{2^{n-1}}$.
  - $\omega_{r_2}(J_2) = 2^{2^{n-2}} \times 2^{2^{n-2}} = 2^{2^{n-1}}$.

  Then, whatever the value of $\alpha \in]0;1[$, SWRPT continues to execute the job J_1 at the date r_2, until its completion at date $r_2 + \alpha$. Starting from the date $r_2 + \alpha$ and until the next release date, r_3, SWRPT executes the job J_2.
- We now show by induction that at the date r_{1+j}, for $1 \leqslant j \leqslant n-1$, the only jobs released earlier than r_{1+j} and whose execution are not yet completed are J_0 with $\rho_{r_{1+j}}(J_0) = 2^{2^{n-2}}$, and J_j with $\rho_{r_{1+j}}(J_j) = \alpha$. We have seen that these properties hold for $j = 1$.

  We now suppose that the properties hold until some value of j included. Then, by induction hypotheses:
  - $\omega_{r_{1+j}}(J_0) = \omega_{r_1}(J_0) = 2^{2^{n-2}} \times 2^{2^n}$.
  - $\omega_{r_{1+j}}(J_j) = \alpha \times 2^{2^{n-j}}$.
  - $\omega_{r_{1+j}}(J_{1+j}) = 2^{2^{n-1-j}} \times 2^{2^{n-1-j}} = 2^{2^{n-j}}$.

  Then, whatever the value of $\alpha \in]0;1[$, SWRPT continues to execute the job J_j at the date r_{1+j}, until its completion at date $r_{1+j} + \alpha$. Starting from the date $r_{1+j} + \alpha$ and until the next release date, r_{2+j}, SWRPT executes the job J_{1+j}. Then the desired properties also hold for $j + 1$.
- Exactly as previously, we can show by induction that at the date r_{n+j}, for $1 \leqslant j \leqslant k-1$, the only jobs released earlier than r_{n+j} and whose execution are not yet completed are J_0 with $\rho_{r_{n+j}}(J_0) = 2^{2^{n-2}}$, and J_{n+j-1} with $\rho_{r_{n+j}}(J_{n+j-1}) = \alpha$.
- We now consider the date r_{n+k+1}.
  - $\omega_{r_{n+k+1}}(J_0) = \omega_{r_1}(J_0) = 2^{2^{n-2}} \times 2^{2^n}$.
  - $\omega_{r_{n+k+1}}(J_{n+k}) = \alpha \times 2^{2^{-k}}$.
  - $\omega_{r_{n+k+1}}(J_{n+k+1}) = 1 \times 1 = 1$.

  Obviously, we want SWRPT to take the wrong decision and to continue to execute job J_{n+k} at date r_{n+k+1}. SWRPT will do that if and only if

$$\alpha \times 2^{2^{-k}} < 1 \Leftrightarrow \alpha < \frac{1}{2^{2^{-k}}}.$$

Therefore, we let $k = \lceil -\log_2(-\log_2 \alpha) \rceil$.

- We can easily show by induction that at the date r_{n+k+j}, for $1 \leqslant j \leqslant l$, the only jobs released earlier than r_{n+k+j} and whose execution are not yet completed are J_0 with $\rho_{r_{n+k+j}}(J_0) = 2^{2^{n-2}}$, and $J_{n+k+j-1}$ with $\rho_{r_{n+k+j}}(J_{n+k+j-1}) = \alpha$.
- Finally, SWRPT executes the job J_{n+k+l} during the time interval $[r_{n+k+l} + \alpha; 1 + r_{n+k+l} + \alpha]$, and then completes the execution of the job J_0 during the time interval $[1 + r_{n+k+l} + \alpha; t_f]$.
- The sum-stretch realized by SWRPT is a bit more complicated to compute than the one realized by SRPT. SWRPT stretches the execution of job J_0 over all the execution of the schedule; job J_1 as a stretch of 1; and the execution of all the other jobs is increased by α. Therefore, the sum-stretch realized by SWRPT on this instance is equal to:

$$\frac{t_f}{2^{2^n}} + 1 + \sum_{j=2}^{n+k} \left(1 + \frac{\alpha}{2^{2^{n-j}}}\right) + l \times \left(1 + \frac{\alpha}{1}\right) = n + k - 1 + l(1+\alpha) + \frac{t_f}{2^{2^n}} + \alpha \sum_{j=2}^{n+k} \frac{1}{2^{2^{n-j}}}.$$

We denote by $\mathcal{R}$ the ratio of the sum-stretch realized by SWRPT on this instance to the optimal sum-stretch. From what precedes, we have:

$$\mathcal{R} \geqslant \frac{n + k - 1 + l\,(1+\alpha) + \frac{t_f}{2^{2^n}} + \alpha \sum_{j=2}^{n+k} \frac{1}{2^{2^{n-j}}}}{n + k + l - 1 + \frac{t_f - \left(2^{2^n} - 2^{2^{n-2}}\right)}{2^{2^{n-1}}}} \geqslant \frac{l\,(1+\alpha)}{n + k + l - 1 + \frac{t_f - \left(2^{2^n} - 2^{2^{n-2}}\right)}{2^{2^{n-1}}}}$$

However, $t_f = l + \sum_{i=0}^{n+k} 2^{2^{n-j}} = l + f(n,k)$. We then have,

$$\mathcal{R} \geqslant \frac{l\,(1+\alpha)}{n + k + l - 1 + \frac{t_f - \left(2^{2^n} - 2^{2^{n-2}}\right)}{2^{2^{n-1}}}} = \frac{l\,(1+\alpha)}{l\left(1 + \frac{1}{2^{2^{n-1}}}\right) + n + k - 1 + \frac{f(k,n) - \left(2^{2^n} - 2^{2^{n-2}}\right)}{2^{2^{n-1}}}}$$

We then choose for n a value large enough to have $\frac{1}{2^{2^{n-1}}} < \frac{\varepsilon}{3(1+\alpha)} = \frac{\varepsilon}{6-\varepsilon}$. α, k, and n are now all defined. Then,

$$\lim_{l \to +\infty} \frac{l\,(1+\alpha)}{l\left(1 + \frac{1}{2^{2^{n-1}}}\right) + n + k - 1 + \frac{f(k,n) - \left(2^{2^n} - 2^{2^{n-2}}\right)}{2^{2^{n-1}}}} = \frac{1+\alpha}{1 + \frac{1}{2^{2^{n-1}}}}.$$

Therefore, we can choose l large enough to have

$$\frac{l\,(1+\alpha)}{l\left(1 + \frac{1}{2^{2^{n-1}}}\right) + n + k - 1 + \frac{f(k,n) - \left(2^{2^n} - 2^{2^{n-2}}\right)}{2^{2^{n-1}}}} \geqslant \frac{1+\alpha}{1 + \frac{1}{2^{2^{n-1}}}} - \frac{\varepsilon}{3}.$$

Then,

$$\mathcal{R} \geqslant \frac{1+\alpha}{1+\frac{1}{2^{2^n-1}}} - \frac{\varepsilon}{3} \geqslant (1+\alpha)\left(1-\frac{1}{2^{2^n-1}}\right) - \frac{\varepsilon}{3} \geqslant (1+\alpha)\left(1-\frac{\varepsilon}{3\,(1+\alpha)}\right) - \frac{\varepsilon}{3} = 1+\alpha-\frac{\varepsilon}{3}-\frac{\varepsilon}{3} = 2-\varepsilon.$$

■

6 Offline Max-Stretch Optimization

Bender, Chakrabarti, and Muthukrishnan [5] 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 $\Omega(n^{1-\varepsilon})$ 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 Section 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 Section 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, Lawler, Lenstra, and Rinnooy Kan [2] presented an $O(n^2)$ algorithm to solve an even more general problem: $\langle 1|\mathit{pmtn}, \mathit{prec}, r_j|f_{\max}\rangle$ (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 (We do not know any reference to this technique who was presented to us by Michael Bender). 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 Section 6.1.4.

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, due to our hypothesis on $\mathcal{F}$, we have:

$$\max_{1\leqslant j\leqslant n} w_j(C_j - r_j) \leqslant \mathcal{F} \quad \Leftrightarrow \quad \forall j \in [1;n],\ w_j(C_j - r_j) \leqslant \mathcal{F} \quad \Leftrightarrow \quad \forall j \in [1;n],\ C_j \leqslant r_j + \mathcal{F}/w_j.$$

Thus, the execution of J_j must be completed before time $\bar{d}_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ 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 $\bar{d}_j$. The problem is then to find a schedule such that each job J_j is executed within its executable time interval $[r_j, \bar{d}_j]$. We consider the set of all job release dates and deadlines: $\{r_1, \ldots, r_n, \bar{d}_1, \ldots, \bar{d}_n\}$. 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:

1. *release date*: job J_j cannot be processed before it is released (Equation (1a));
2. *deadline*: job J_j cannot be processed after its deadline (Equation (1b));
3. *resource usage*: during a time interval, a machine cannot be used longer than the duration of this time interval (Equation (1c));
4. *job completion*: each job must be processed to completion (Equation (1d)).

$$\begin{cases} \text{(1a)} & \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(1b)} & \forall i, \forall j, \forall t, \quad \bar{d}_j \leqslant \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(1c)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leqslant \sup I_t - \inf I_t \\ \text{(1d)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \end{cases} \tag{1}$$

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 smaller 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, $r_1, \ldots, r_n, \bar{d}_1(\mathcal{F}), \ldots, \bar{d}_n(\mathcal{F})$, 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 $\{r_1, \ldots, r_n, \bar{d}_1(\mathcal{F}), \ldots, \bar{d}_n(\mathcal{F})\}$ 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 $[\mathcal{F}_1, \mathcal{F}_2]$ ($\mathcal{F}_1 \leqslant \mathcal{F} \leqslant \mathcal{F}_2$), and with the minimization of $\mathcal{F}$ as the objective. This gives us System (2).

$$
\begin{array}{l}
\text{MINIMIZE} \quad \mathcal{F} \quad , \\
\text{UNDER THE CONSTRAINTS} \\
\left\{ \begin{array}{lll}
\text{(2a)} & \mathcal{F}_1 \leqslant \mathcal{F} \leqslant \mathcal{F}_2 & \\
\text{(2b)} & \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 & \\
\text{(2c)} & \forall i, \forall j, \forall t, \quad \bar{d}_j \leqslant \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 & \\
\text{(2d)} & \forall t, \forall i, \quad \displaystyle\sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leqslant \sup I_t - \inf I_t & \\
\text{(2e)} & \forall j, \quad \displaystyle\sum_t \sum_i \alpha_{i,j}^{(t)} = 1 &
\end{array} \right.
\end{array}
\tag{2}
$$

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*[1]. 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, $1 \leqslant n_{\text{q}} \leqslant n^2 - n$. We denote by $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_{\text{q}}}$ the milestones ordered by increasing values. To solve our problem we just need to perform a

[1] Labetoulle, Lawler, Lenstra, and Rinnooy Kan [19] call such a value a "critical trial value".

binary search on the set of milestones $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_q}$, each time checking whether System (2) has a solution in the objective interval $[\mathcal{F}_i, \mathcal{F}_{i+1}]$ (except for $i = n_q$ in which case we search for a solution in the range $[\mathcal{F}_{n_q}, +\infty[$). There is a polynomial number of milestones and System (2) can be solved in polynomial time. Therefore:

Theorem 9. *$\langle R|r_j; div|\max w_j F_j\rangle$ is polynomial : minimizing the maximum weighted flow is a polynomial problem, in the divisible load model.*

6.1.4 A Network Flow Approach for Uniform Machines

In section 6.1.1, we presented Linear program 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, 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, $1 \leqslant j \leqslant n$;
- One vertex (I_t, M_i) for each couple made of a time interval I_t, $1 \leqslant t \leqslant n_{\text{int}}$, and of a machine M_i, $1 \leqslant i \leqslant m$.

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., $r_j \leqslant \inf I_t$ and $\sup t \leqslant \bar{d}_j$). 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 8 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

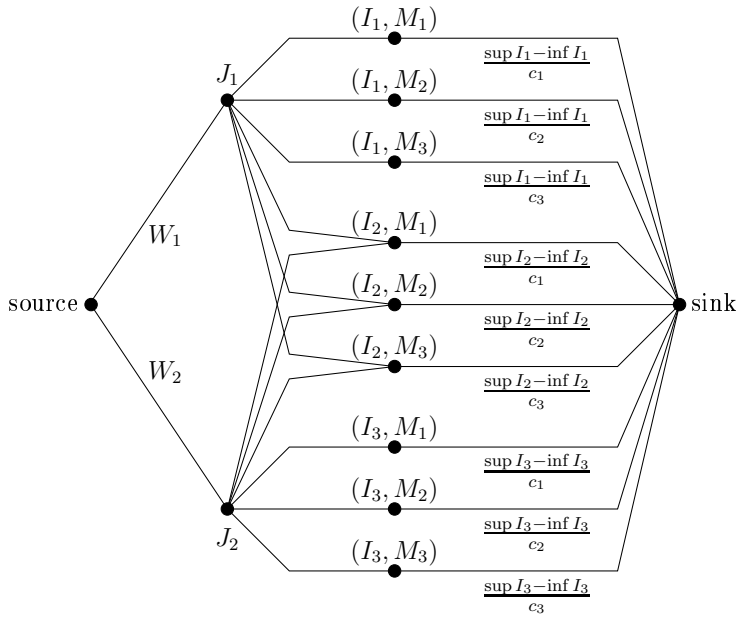


Figure 8: Graph used to check, on uniform machines and through network flow maximization, whether there exists a schedule of a given maximum weighted flow. This example has two jobs, three machines, and three time intervals defined by the epochal times $r_1 < r_2 < \bar{d}_1 < \bar{d}_2$.

example of Figure 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 when there are no problems of 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 [20] 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 [16], Lawler and Labetoulle [20] present a scheme to build in polynomial-time a preemptive schedule of makespan $\mathcal{C}$ for a set of jobs

$J_1, ..., J_n$ 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.

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

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 Section 6.1.1, we use this objective value to define for each job J_j a deadline $\bar{d}_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$. 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 $I_1, \ldots, I_{n_{\text{int}}(\mathcal{F})}$. 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:

1. each job must be processed to completion (Equation (4a) which corresponds to Equation (3a));
2. 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));
3. the utilization of a machine during a time interval cannot exceed its capacity (Equation (4c) which corresponds to Equation (3c));
4. the processing of a job cannot start before it is released (Equation (4d));
5. a job must be processed before its deadline (Equation (4e)).

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

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 [20], 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 Section 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:

Theorem 10. $\langle R|r_j; pmtn|\max w_j F_j\rangle$ *is polynomial : minimizing the maximum weighted flow, with preemption, is a polynomial problem.*

7 Offline Max-stretch Optimization and Pareto Optimality

7.1 Pareto-Optimality

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.

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). *C is Pareto-optimal if and only if:*

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

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 strictly a 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:

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

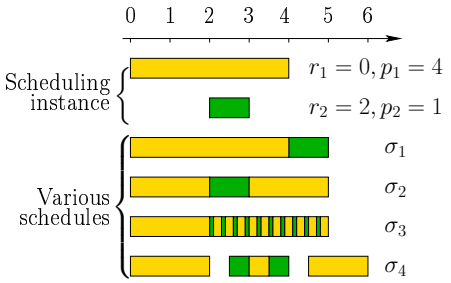
Figure 9 depicts on each subfigure, for a simple scheduling instance the various cost sets associated to the completion time, flow time and stretch metrics. The dashed-dotted line is the optimal isoline for the considered max-based metric ($\max_j C_j$ for Figure 9(b), $\max_j F_j$ for Figure 9(c), and $\max_j S_j$ for Figure 9(d)). 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 [8, 25]) 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.

7.2 Heuristic Pareto Minimization of Max-Stretch on One Machine

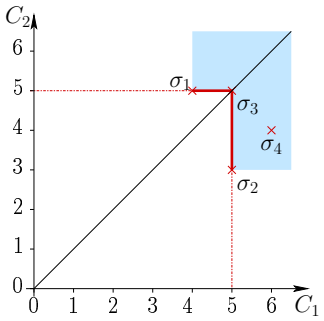
Algorithm 1 is an obvious algorithm which recursively tries to minimize the stretch of jobs: first it minimizes the max-stretch, then the number of jobs whose stretch is equal to the max-stretch, then the maximum stretch of the other jobs, and so on. We show that in some cases Algorithm 1 produces Pareto optimal schedules for stretch minimization.

Algorithm 1: Heuristic Pareto minimization of max-stretch on one machine.

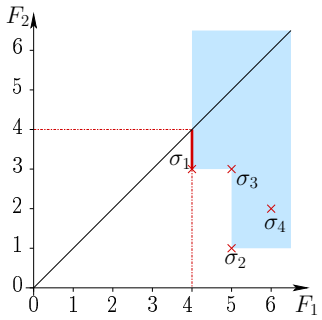
FixedStretch $\leftarrow \emptyset$
FreeStretch $\leftarrow \{J_1, ..., J_n\}$
while *FreeStretch* $\neq \emptyset$ **do**
    Compute the minimum maximum stretch $\mathcal{S}$ of the jobs in *FreeStretch* taking into account that for any job J_j such that $(J_j, \mathcal{S}_j) \in$ *FixedStretch*, J_j has exactly a stretch of $\mathcal{S}_j$.
    foreach $J_j \in$ *FreeStretch* **do**
        Let $\bar{d}_j \leftarrow r_j + \mathcal{S} \times p_j$
    foreach $(J_j, \mathcal{S}_j) \in$ *FixedStretch* **do**
        Let $\bar{d}_j \leftarrow r_j + \mathcal{S}_j \times p_j$
    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
    foreach $J_j \in$ *FreeStretch* **do**
        if $C_j = \bar{d}_j$ **then**
            FreeStretch $\leftarrow$ *FreeStretch* $\setminus \{J_j\}$
            FixedStretch $\leftarrow$ *FixedStretch* $\cup \{(J_j, \mathcal{S})\}$
foreach $(J_j, \mathcal{S}_j) \in$ *FixedStretch* **do**
    Let $\bar{d}_j \leftarrow r_j + \mathcal{S}_j \times p_j$
Schedule Earliest Deadline First (EDF) all the jobs



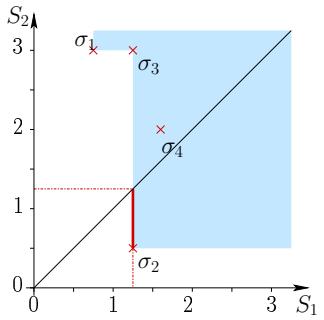
(a) Scheduling instance and various feasible schedules.



(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 $\max_j S_j$.

Figure 9: Most optimal solutions to max-based metrics are not Pareto-optimal.

Theorem 11. *Algorithm 1 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 and/or 8, are equal.*

Proof. We will prove the correction of Algorithm 1 in two steps. First we will show that the algorithm always terminates. Then we will show that it produces an optimal schedule if at no iteration of the while loop there are two jobs whose deadlines, defined at Steps 6 and/or 8, are equal.

1. We first show that all algorithms steps are feasible and that the algorithms terminates.

   Initially, *FixedStretch* is empty and the max-stretch minimization of Step 4 is absolutely equivalent to what we have done in Section 6 and is thus trouble-less. Later on,

the problem solved at Step 4 of a given iteration of the while loop has, as a solution, the schedule found by EDF at the previous iteration of the while loop, because of the way we fix the stretch values for jobs in *FixedStretch*. Therefore, the problem at Step 4 is always feasible.

The algorithm terminates as the size of *FreeStretch* strictly decreases with each iteration of the while loop. Indeed, at Step 11 the condition is true for at least one job at each iteration of the while loop. Otherwise, using EDF we would have found a schedule under which the max-stretch of the jobs in *FreeStretch* would be strictly smaller than the value found at Step 4 which is impossible.

2. We now suppose that, whatever the iteration of the while loop, there does not exist two jobs whose deadlines, defined at Steps 6 and/or 8, are equal. We then prove by induction that the computed schedule is Pareto optimal.

   Initially, the set *FixedStretch* is empty and Step 4 computes the minimum max-stretch $\mathcal{S}^*$ achievable. The only question we have to answer is thus: is the number of jobs whose stretch is fixed to $\mathcal{S}^*$ minimal? We will prove a stronger result: if our algorithm sets the stretch of a job J_j to $\mathcal{S}^*$, then in all schedules whose max-stretch is less than or equal to $\mathcal{S}^*$, the stretch of J_j is actually equal to $\mathcal{S}^*$. We prove this by contradiction. We thus assume that there exists some schedule Θ whose max-stretch is less than or equal to $\mathcal{S}^*$, and under which the stretch of J_j is equal to $\mathcal{S}_j^{(\Theta)} < \mathcal{S}^*$. Then we define the following instance of deadline scheduling: the deadline of any job $J_i \neq J_j$ is the same as under our algorithm, i.e., $\bar{d}_i = r_i + \mathcal{S}^* \times p_i$, and the deadline of J_i is equal to $\bar{d}_i = r_i + (\mathcal{S}^* - \varepsilon) \times p_i$, where $\varepsilon > 0$ is chosen smaller enough such that $\mathcal{S}_j^{(\Theta)} \leqslant \mathcal{S}^* - \varepsilon$, and such that the order of the deadlines we just defined, sorted by non-decreasing values, is the same as for the ones defined by Step 6 (which is the order found when $\varepsilon = 0$). Such an ε exists *because* we have made the hypothesis that no two jobs have the same deadline at the first iteration of the while loop. Our instance of deadline scheduling is feasible as it admits Θ as a solution. Now, we schedule our instance using EDF. As EDF always finds a valid schedule if one exists [13], it finds a solution for this instance. EDF schedules the jobs under this instance exactly as it did in Algorithm 1, as the order of the deadlines did not change, and thus finds the same stretches. Therefore, EDF finds a stretch of $\mathcal{S}^*$ for J_j which is impossible looking at the definition of J_j's deadline, $\bar{d}_j$. Hence a contradiction.

   The general case of the induction is proven the same way as it also relies on the facts that 1) each time a stretch is fixed, it is the minimal maximum due to Step 4; and 2) EDF succeeds whenever there is a valid schedule. ■

We conjecture that Algorithm 1 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.

7.3 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.

Algorithm 2 presents the solution we propose for the general case. The solution for uni-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 *FreeStretch* job. 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 first show that Algorithm 2 is correct. Then we come back on Step 15, which is not fully defined.

Lemma 3. *Algorithm 2 produces a valid schedule.*

Proof. The proof of correction of Algorithm 2 follows from the proof of correction for Algorithm 1, except for the loop at Step 10. We have therefore to prove two properties: 1) System 5 has a null solution for deadline d if and only if, whatever the schedule, there exists a job J_j such that $C_j = \bar{d}_j = d$ (we then say d is a "tight" deadline); 2) there exists a valid schedule under which, whatever the deadline d which is not tight, there is no job J_j such that $C_j = \bar{d}_j = d$.

We now prove the first property. Suppose $d \in \mathcal{D}$ is not a tight deadline. Then there exists a schedule Θ such that all jobs whose deadline is d complete strictly before the date d. We consider the time interval I_{t_d} which ends at date d (see Section 6), and any processor P_i. As no jobs whose deadline is d completes at date d, right before that date either:

1. P_i is idle, and then:
$$\sup I_{t_d} - \inf I_{t_d} > \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \geqslant \sum_{j|\bar{d}_j=d} \alpha_{i,j}^{(t)}.p_{i,j}.$$
2. P_i processes a job whose deadline is strictly greater than d, and then:
$$\sum_{j|\bar{d}_j=d} \alpha_{i,j}^{(t)}.p_{i,j} < \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leqslant \sup I_{t_d} - \inf I_{t_d}.$$

In all cases:
$$\sup I_{t_d} - \inf I_{t_d} - \sum_{j|\bar{d}_j=d} \alpha_{i,j}^{(t)}.p_{i,j} > 0$$

Algorithm 2: Heuristic Pareto minimization of max weighted flow.

$FixedStretch \leftarrow \emptyset$
$FreeStretch \leftarrow \{J_1, ..., J_n\}$
while $FreeStretch \neq \emptyset$ **do**
  Compute the minimum max weighted flow $\mathcal{S}$ 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 $\mathcal{S}_j$
  foreach $J_j \in FreeStretch$ **do**
    $\bar{d}_j \leftarrow r_j + \mathcal{S} \times p_j$
  foreach $(J_j, \mathcal{S}_j) \in FixedStretch$ **do**
    $\bar{d}_j \leftarrow r_j + \mathcal{S}_j \times p_j$
  $\mathcal{D} \leftarrow \{\bar{d}_j \mid S_j \in FreeStretch\}$
  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)

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

    if $\delta = 0$ **then**
      $S_d \leftarrow \{J_j \in FreeStretch \mid \bar{d}_j = d\}$
      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**
        $FreeStretch \leftarrow FreeStretch \setminus \{J_j\}$
        $FixedStretch \leftarrow FixedStretch \cup \{(J_j, \mathcal{S})\}$
foreach $(J_j, \mathcal{S}_j) \in FixedStretch$ **do**
  $\bar{d}_j \leftarrow r_j + \mathcal{S}_j \times p_j$
Build a schedule according to the solution of Linear Program 1.

Thus, we can pick for δ the strictly positive value:

$$\min_i \left(\sup I_{t_d} - \inf I_{t_d} - \sum_{j|\bar{d}_j=d} \alpha_{i,j}^{(t)}.p_{i,j} \right).$$

Therefore, if $\delta = 0$, d is a tight schedule.

Conversely, if $\delta > 0$, we take any solution to System (5), and then, on each processor, and during each time interval, we schedule earliest deadline first the fractions $\alpha_{i,j}^{(t)}$. As $\delta > 0$, whatever the processor, $\inf I_{t_d} + \sum_{j|\bar{d}_j=d} \alpha_{i,j}^{(t)}.p_{i,j} < \sup I_{t_d}$ and, thus, all jobs those deadlines are d are completed strictly before the date d.

We now prove the second property. Let d_1 and d_2 be two deadlines in $\mathcal{D}$. Let Θ_1 and Θ_2 be two schedules such that under Θ_1 (resp. Θ_2) no job J_j is such that $C_j = \bar{d}_j = d_1$ (resp. $C_j = \bar{d}_j = d_2$). We denote by $\alpha_{i,j}^{(t,1)}$ (resp. $\alpha_{i,j}^{(t,2)}$) the fraction of job J_j processed on processor P_i during the time interval I_t under the schedule Θ_1 (resp. Θ_2). We then define a third schedule, Θ_3, by $\alpha_{i,j}^{(t,3)} = \frac{1}{2}(\alpha_{i,j}^{(t,1)} + \alpha_{i,j}^{(t,2)})$, and by scheduling, on each processor, and during each interval, the fractions Earliest Deadline First. One can easily check that Θ_3 is a valid schedule and that under Θ_3, there is no job J_j such that $C_j = \bar{d}_j = d_1$ or $C_j = \bar{d}_j = d_2$. An immediate induction gives us the desired property. ■

Step 15 does not explicit how the set "S'_d" 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, as shown by the proof of the next theorem which states the complexity of the general max-flow minimization, and thus of the general case.

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

As we do not have any release dates in the above theorem, we in fact prove that $\langle R|div|C_{\max} Pareto\rangle$, 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.

Proof. This result is proved with a reduction from MINIMUM HITTING SET [15].

Let us consider any instance $\mathcal{I}_1$ of MINIMUM HITTING SET. $\mathcal{I}_1$ is defined by a collection $C = \{S_1, ..., S_{|C|}\}$ of subsets of a finite set S and by an integer K. The question is: is there a subset S' of S, such that $|S'| \leqslant K$ and such that S' contains at least one element from each subset in C: for each $i \in [1, |C|], S_i \cap S' \neq \emptyset$. Without loss of generality we assume that $S = \cup_i S_i$.

From instance $\mathcal{I}_1$ of MINIMUM HITTING SET, we now build an instance $\mathcal{I}_2$ of our problem. $\mathcal{I}_2$ is made of $n = |S|$ jobs and we identify the jobs J_1, ..., $J_{|S|}$ with the elements x_1, ..., $x_{|S|}$ of S. The size W_j of job J_j is equal to the number of subsets containing x_j : $W_J = |\{S_i \in C \mid x_j \in S_i\}|$. We will have to schedule these jobs on $m = |C|$ processors

and we identify the processors with the subsets S_1, ..., $S_{|C|}$. We define the computational characteristics of the processors as follows:

$$p_{i,j} = \begin{cases} \frac{1}{|S_i|} & \text{if } x_j \in S_i, \\ \infty & \text{otherwise.} \end{cases}.$$

Here the question is: is there a schedule for which the number of jobs whose flow is equal to the optimal max-flow is less than or equal to K?

We first remark that the optimal maximum flow is equal to 1. Indeed, the total load to be processed is $\sum_j W_j = \sum_j |\{S_i \in C \mid x_j \in S_i\}| = \sum_i |S_i|$ and, at best, processor P_i can process a load of size $|S_i|$ during a unit of time. Therefore the optimal max-flow is greater than or equal to 1. A max-flow of 1 is realized by any schedule under which processor P_i devotes a fraction $\frac{1}{|S_i|}$ of the time interval $[0, 1]$ to any job J_j such that $x_j \in S_i$. Indeed, under such a schedule, the share of job J_j processed during the time interval $[0, 1]$ by processor P_i such that $x_j \in S_i$ is equal to 1. Therefore, the overall share of job J_j process during that time interval is equal to $\sum_{i|x_j \in S_i} 1 = |\{S_i \in C \mid x_j \in S_i\}| = W_j$.

Furthermore, this proof shows that if any processor is (at least) partially idle during the time interval $[0, 1]$ then the max-flow achieved will be strictly greater than one. Therefore, under any schedule achieving the optimal max-flow there is, on each processor, a job which is run until the date 1, and thus which has a max-flow of 1. The set of jobs whose max-flow is one in $\mathcal{I}_2$ then equivalently defines a hitting set of S in $\mathcal{I}_1$. ■

MINIMUM HITTING SET is equivalent to MINIMUM SET COVER [14]. 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 $1 + \ln |S|$ [18, 32], 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 2 produces an optimal schedule. Therefore:

Theorem 13. *Algorithm 2 produces a Pareto optimal schedule for max-stretch 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 2 will always produce optimal schedules in practice. (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, Chakrabarti, and Muthukrishnan [5]. Then we present the two competitive algorithms that have previously been proposed in the literature [5, 6]. 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. *For three lengths of jobs, there is no* $\frac{1}{2}\Delta^{\sqrt{2}-1}$*-competitive preemptive online algorithm minimizing max-stretch.*

This result is an improvement from the bound of $\frac{1}{2}\Delta^{\frac{1}{3}}$ established by Bender, Chakrabarti, and Muthukrishnan [5]. 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 $r_1 = r_2 = 0$.

If these two jobs were the only ones received by the processor, the optimal max-stretch would be 2.

Phase 2: Starting at time $2\delta - k$, and every k time units, arrives a job of size k (with $k < \delta$). There are x such jobs. In other words, for $1 \leqslant j \leqslant x$, 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 as, under such a scheduling, J_{2+j} is executed during the interval $[r_j + k, r_j + 2k]$.

The 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: $\frac{C_1 - r_1}{p_1} \leqslant \frac{1}{2}\Delta^{\sqrt{2}-1} \times 2$ (we have the 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) [2]. Therefore, the constraint on C_1 can

[2] 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.

be rewritten:

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

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 each of the jobs J_3, ..., J_{2+x} at its release date. To forbid such a behavior, we just have to choose x, the number of jobs of size k, to be large enough for algorithm $\mathcal{A}$ 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, ..., 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, $2\delta + (x-1)k > \frac{\delta^{\sqrt{2}}}{k^{\sqrt{2}-1}}$ and 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 $1 \leqslant j \leqslant y$, job J_{2+x+j} arrives at time $r_{2+x+j} = 2\delta + (x-1)k + (j-1)$ and is of size $p_{2+x+j} = 1$.

Let us evaluate the best max-stretch that can be achieved by any schedule. A possible schedule would be:

1. Execute J_1 during the time interval $[0, \delta]$, hence a stretch of 1 for J_1.
2. Execute a chunk of J_2 during the time interval $[\delta, 2\delta - k]$.
3. Execute each of the jobs of size k and 1 at the time of its arrival, hence a stretch of 1 for each of them.
4. Complete the execution of J_2 during the time interval $[2\delta + (x-1)k + y, 2\delta + xk + y]$, hence a stretch of $\frac{2\delta+xk+y}{\delta}$ for J_2.

Therefore, the optimal max-stretch is less than or equal to: $\frac{2\delta+xk+y}{\delta}$.

Let us now evaluate the stretch realized by algorithm $\mathcal{A}$. Its schedule can end at the earliest at time $2\delta + xk + y$. As we have forced algorithm $\mathcal{A}$ to end the execution of C_1 and C_2 by the date $2\delta + (x-1)k$, the schedule can end either with a job of size 1 or with one of size k:

1. If the last job is of size 1, we denote it J_{2+x+j}. Then, its stretch satisfies:

$$\text{stretch}(J_{2+x+j}) \geqslant \frac{(2\delta + xk + y) - (2\delta + (x-1)k + (j-1))}{1} = k + (y-j) + 1 \geqslant k+1.$$

2. If the last job is of size k, we denote it J_{2+j}. Then, its stretch satisfies:

$$\text{stretch}(J_{2+j}) \geqslant \frac{(2\delta + xk + y) - (2\delta + (j-2)k)}{k} = (x - j + 2) + \frac{y}{k} \geqslant 2 + \frac{y}{k}.$$

Therefore, the max-stretch that algorithm $\mathcal{A}$ can achieve is greater than or equal to

$$\min\left\{k+1, 2+\frac{y}{k}\right\}.$$

We then define y to maximize this value. We thus pick for y the smallest value that enable us to have $2 + \frac{y}{k} \geqslant k+1$, i.e., we choose $y = \lceil k(k-1) \rceil$.

So far, algorithm $\mathcal{A}$ achieves a max-stretch greater than or equal to $k+1$, when the optimal max-stretch is less than or equal to $\frac{2\delta+xk+y}{\delta}$. Thus, if we can choose k such that:

$$k+1 > \frac{1}{2}\Delta^{\sqrt{2}-1}\frac{2\delta + xk + y}{\delta},$$

we will have reached the desire contradiction. Here, we have $\Delta = \frac{\delta}{1} = \delta$. We pick $k = \delta^{2-\sqrt{2}}$. We then have:

$$\begin{aligned}
(k+1) - \frac{1}{2}\delta^{\sqrt{2}-1}\frac{2\delta + xk + y}{\delta} &= k + 1 - \frac{1}{2}\delta^{\sqrt{2}-1}\left(2 + \left\lfloor 2 + \left(\frac{\delta}{k}\right)^{\sqrt{2}} - \frac{2\delta}{k}\right\rfloor \frac{k}{\delta} + \frac{\lceil k(k-1)\rceil}{\delta}\right) \\
&> k + 1 - \frac{1}{2}\delta^{\sqrt{2}-1}\left(2 + \left(2 + \left(\frac{\delta}{k}\right)^{\sqrt{2}} - \frac{2\delta}{k}\right)\frac{k}{\delta} + \frac{1 + (k(k-1))}{\delta}\right) \\
&= \delta^{2-\sqrt{2}} + 1 - \frac{1}{2}\delta^{\sqrt{2}-1}\left(2\delta^{1-\sqrt{2}} + \delta^{3-2\sqrt{2}} + \delta^{5-4\sqrt{2}} - \delta^{1-\sqrt{2}} + \delta^{-1}\right) \\
&= \frac{1}{2} + \frac{1}{2}\delta^{2-\sqrt{2}}\left(1 - \delta^{2-2\sqrt{2}} - \delta^{2\sqrt{2}-4}\right) \\
&> 0
\end{aligned}$$

The last inequality comes from the fact that $2 - 2\sqrt{2} < 0$ and $2\sqrt{2} - 4 < 0$, and it holds when δ is large enough (e.g., $\delta > 2^{\frac{1}{2(\sqrt{2}-1)}}$). ■

8.2 Competitive Online Heuristics

We have already seen in Section 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, Muthukrishnan, and Rajaraman [6] defined, for any job J_j, a pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

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

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, Chakrabarti, and Muthukrishnan [5] 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: $\bar{d}_j(\mathcal{F}) = r_j + \alpha \times \mathcal{S}^*/p_j$ 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 Section 10. 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 2 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).

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

UNDER THE CONSTRAINTS

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

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 Section 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 section 10.

9 Summary of Complexity Results

Table 1 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 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 $\langle 1|r_j, pmtn|\sum S_j\rangle$ even if (as we already have mentioned in Section 5.1) Polynomial Time Approximation Scheme (PTAS) have been proposed for this problem.

10 Simulations

To evaluate the efficacy of various scheduling strategies when optimizing stretch-based metrics, we implemented a simulator using the SimGrid toolkit [21], 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 Section 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

	$model = \emptyset$	$model = pmtn$	$model = div$
$\langle 1 \mid r_j; model \mid \max w_j F_j \rangle$	NP([5])	$\downarrow$	$\downarrow$
$\langle P \mid r_j; model \mid \max w_j F_j \rangle$	$\uparrow$	$\downarrow$	$\downarrow$
$\langle Q \mid r_j; model \mid \max w_j F_j \rangle$	$\uparrow$	$\downarrow$	$\downarrow$
$\langle R \mid r_j; model \mid \max w_j F_j \rangle$	$\uparrow$	P(Lin. Prog. Sec. 6.2)	P(Lin. Prog. Sec. 6.1)
$\langle 1 \mid r_j; model \mid \sum F_j \rangle$	NP([24])	P(SRPT [1])	$\downarrow$
$\langle P \mid r_j; model \mid \sum F_j \rangle$	$\uparrow$	NP(Numerical-3DM [3])	$\downarrow$
$\langle Q \mid r_j; model \mid \sum F_j \rangle$	$\uparrow$	$\uparrow$	P(SRPT + Sec. 2.3)
$\langle R \mid r_j; model \mid \sum F_j \rangle$	$\uparrow$	$\uparrow$	NP(3DM, Sec. 4)
$\langle 1 \mid r_j; model \mid \sum S_j \rangle$	NP(Sec. 5.1)	?	?
$\langle P \mid r_j; model \mid \sum S_j \rangle$	$\uparrow$	?	?
$\langle Q \mid r_j; model \mid \sum S_j \rangle$	$\uparrow$	?	?
$\langle R \mid r_j; model \mid \sum S_j \rangle$	$\uparrow$	?	NP(3DM, Sec. 5.1)
$\langle 1 \mid r_j; model \mid \sum w_j F_j \rangle$	NP([24])	NP(Numerical-3DM [19])	$\nwarrow$
$\langle P \mid r_j; model \mid \sum w_j F_j \rangle$	$\uparrow$	$\uparrow$	$\uparrow$
$\langle Q \mid r_j; model \mid \sum w_j F_j \rangle$	$\uparrow$	$\uparrow$	$\uparrow$
$\langle R \mid r_j; model \mid \sum w_j F_j \rangle$	$\uparrow$	$\uparrow$	$\uparrow$

Table 1: Summary of complexity results.

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 parameterizations of the platform and application models.

10.1 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.

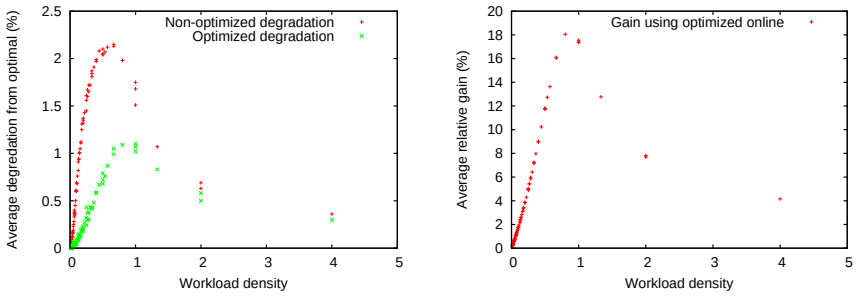
We define a *simulation configuration* as a set of specific values for each of these six properties. Once defined, concrete *simulation instances* are constructed by realizing random series for any random variables in the system. In particular, two models are created for each instance: a platform model 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 inter-arrival 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 Section 10.3.

10.2 Optimization of the Online Heuristic

To motivate the variants of our online heuristic described in Section 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. Figures 10(a) and 10(b) present 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).



(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

Figure 10: Comparison of the optimized and non-optimized versions of the online heuristic.

10.3 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 Section 6 that solves the optimal max-stretch problem. Two versions of the online heuristic are also implemented, designated as ONLINE and ONLINE-EGDF. Next, we consider the SWRPT, SRPT, and SPT heuristics discussed in Section 5. Then, we consider the two online heuristics proposed by Bender *et al.* that were briefly described in Section 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 Section 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

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0001	1.0141	1.5503	0.2940	2.9472
ONLINE:	1.0041	0.0221	3.9853	1.0495	0.0398	1.4158
ONLINE-EGDF:	1.0387	0.0681	1.7772	1.0025	0.0043	1.0935
SWRPT:	1.0515	0.0878	1.9632	1.0002	0.0010	1.0426
SRPT:	1.0688	0.1046	1.9319	1.0049	0.0050	1.1115
SPT:	1.0778	0.1229	2.3820	1.0022	0.0039	1.1814
BENDER98[3]:	1.0667	0.1228	2.4877	1.0037	0.0068	1.1701
BENDER02:	3.6651	3.0509	26.4899	1.2074	0.2532	7.1756
FCFS-DIV:	7.0806	8.1259	78.0723	1.3856	0.5911	18.2120
MCT:	34.1058	21.3104	144.1156	52.6756	36.9028	161.9829
RAND:	6.7452	8.8499	133.7370	1.2413	0.3723	10.3411

Table 2: Aggregate statistics over all 162 platform/application configurations.

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 uni-processor heuristics (SWRPT, SRPT, SPT, and Bender *et al.*'s) are extended to the multi-processor case using the strategy previously described in Section 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 [23]. 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; and

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 2 presents the aggregate results from these simulations; finer-grained results based on various partitionings of the data may be found in the Appendix A.

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 34 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 ten 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 7.0 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 of the two online heuristics we propose is consistently near-optimal (within 4‰ on average) for max-stretch optimization. The second heuristic, ONLINE-EGDF, actually achieves consistently good sum-stretch (within 2‰ 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 the heuristic ignores a significant portion of the fine-tuned schedule generated by the linear program designed to optimize the max-stretch.

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 0.2‰ within the best observed sum-stretch, and attaining a sum-stretch within 5% 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 n^{th} 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 2 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

[3] BENDER98 results are limited to 3-cluster platforms, due to prohibitive overhead costs (discussed in Section 10.3).

	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: Aggregate statistics for a single machine for all application configurations.

to unrelated machines configurations (see Section 2.3). Indeed, similar observations can be done when restricting to single-machine configurations (see Table 3).

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 discrimating objective for our problem. Indeed, it looks as if, whatever the policy, any list-scheduling heuristic delivers good performance for this metric.

Finally, we note the anomalous result that the optimal algorithm is occasionally beaten (in all cases by one of our online heuristic); clearly this indicates an error in the solution of the optimal max-stretch problem. Our analysis suggests that this is a floating-point precision problem that arises when very fine variations in values of $\mathcal{F}$ result in different orderings of the epochal times. We are considering potential solutions to the problem, such as scaling the linear program variables such that precision errors between epochal times may be avoided.

11 Conclusion

Our initial goal was to minimize the maximum stretch. We have presented a polynomial-time algorithm to solve this problem off-line. We have also proposed some heuristics to solve this problem on-line. 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. However, some questions remain open, like:

- What is the complexity of $\langle 1|pmtn; r_j|\sum S_j\rangle$, the sum-stretch minimization on one machine with preemption ?
- What is the complexity of $\langle R|div; r_j|ParetoS_{\max}\rangle$, 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 [5, 4], 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 therefore: what is the competitive ratio of Processor Sharing for max-stretch ?

Acknowledgments

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.

References

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

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

[3] Philippe Baptiste, Peter Brucker, Marek Chrobak, Christoph Durr, Svetlana A. Kravchenko, and Francis Sourd. The complexity of mean flow time scheduling problems with release times, 2006. Available at `http://arxiv.org/abs/cs/0605078`.

[4] Michael A. Bender. *New Algorithms and Metrics for Scheduling.* PhD thesis, Harvard University, May 1998.

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

[6] Michael A. Bender, S. Muthukrishnan, and Rajmohan Rajaraman. Improved algorithms for stretch scheduling. In *SODA '02: Proceedings of the thirteenth annual ACM-SIAM*

symposium on Discrete algorithms, pages 762–771, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

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

[8] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.

[9] Christophe Blanchet, Christophe Combet, Christophe Geourjon, and Gilbert Deléage. MPSA: Integrated System for Multiple Protein Sequence Analysis with client/server capabilities. *Bioinformatics*, 16(3):286–287, 2000.

[10] R. C. Braun, Kevin T. Pedretti, Thomas L. Casavant, Todd E. Scheetz, C. L. Birkett, and Chad A. Roberts. Parallelization of local BLAST service on workstation clusters. *Future Generation Computer Systems*, 17(6):745–754, 2001.

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

[12] Aaron E. Darling, Lucas Carey, and Wu chun Feng. The Design, Implementation, and Evaluation of mpiBLAST. In *Proceedings of ClusterWorld 2003*, 2003.

[13] M. L. Dertouzos. Control robotics: the procedural control of physical processes. In *Proceedings of IFIP Congress*, pages 897–813, 1974.

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

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.

[16] Teofilo Gonzalez and Sartaj Sahni. Open shop scheduling to minimize finish time. *J. ACM*, 23(4):665–679, 1976.

[17] GriPPS webpage at `http://gripps.ibcp.fr/`, 2005.

[18] D. S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

[19] Jacques Labetoulle, Eugene L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In W. R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 245–261. Academic Press, 1984.

[20] Eugene L. Lawler and Jacques Labetoulle. On preemptive scheduling of unrelated parallel processors by linear programming. *Journal of the Association for Computing Machinery*, 25(4):612–619, 1978.

[21] Arnaud Legrand, Loris Marchal, and Henri Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Proceedings of the 3rd IEEE Symposium on Cluster Computing and the Grid*, 2003.

[22] Arnaud Legrand, Alan Su, and Frédéric Vivien. 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.

[23] Arnaud Legrand, Alan Su, and Frédéric Vivien. 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. IEEE Computer Society Press.

[24] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[25] Laurent Massoulié and James Roberts. Bandwidth sharing: Objectives and algorithms. *Transactions on Networking*, 10(3):320–328, june 2002.

[26] Mathematica 5.2. `http://www.wolfram.com/`.

[27] Nicole Megow. Performance analysis of on-line algorithms in machine scheduling. Diplomarbeit, Technische Universität Berlin, April 2002.

[28] Perry L. Miller, Prakash M. Nadkarni, and N. M. Carriero. 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, 1991.

[29] S. Muthukrishnan, Rajmohan Rajaraman, Anthony Shaheen, and Johannes Gehrke. Online scheduling to minimize average stretch. In *IEEE Symposium on Foundations of Computer Science*, pages 433–442, 1999.

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

[31] René Sitters. Complexity of preemptive minsum scheduling on unrelated parallel machines. *Journal of Algorithms*, 57(1):37–48, September 2005.

[32] Petr Slavík. 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*, pages 435–441, New York, NY, USA, 1996. ACM Press.

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

A Detailed simulation results

In each of the following sections, we show the aggregate results when the value of one of the parameters is fixed. Remember that we were only able to run the BENDER98 heuristic on platforms containing 3 clusters, as this heuristic is too computationally intensive.

A.1 Platform size

A.1.1 Platforms with 3 clusters

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0000	1.0000	1.3250	0.2498	2.4069
ONLINE:	1.0032	0.0155	1.4115	1.0344	0.0345	1.3726
ONLINE-EGDF:	1.0319	0.0701	1.7086	1.0024	0.0055	1.0935
SWRPT:	1.0358	0.0771	1.8640	1.0004	0.0015	1.0426
SRPT:	1.0618	0.1110	1.9075	1.0050	0.0067	1.1115
SPT:	1.0645	0.1222	2.3820	1.0027	0.0053	1.1814
BENDER98:	1.0667	0.1228	2.4877	1.0037	0.0068	1.1701
BENDER02:	3.5760	3.2157	23.6092	1.2250	0.3121	7.1756
FCFS-DIV:	7.4381	8.9431	78.0723	1.4789	0.7843	18.2120
MCT:	13.8534	4.3962	33.5998	17.7493	4.7573	30.0000
RAND:	6.7376	9.3419	110.6094	1.3057	0.5162	10.3411

A.1.2 Platforms with 10 clusters

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0001	1.0141	1.6130	0.2441	2.6638
ONLINE:	1.0043	0.0138	1.1924	1.0556	0.0400	1.4158
ONLINE-EGDF:	1.0417	0.0688	1.7772	1.0026	0.0038	1.0804
SWRPT:	1.0563	0.0901	1.9632	1.0002	0.0006	1.0155
SRPT:	1.0718	0.1030	1.9319	1.0049	0.0042	1.0482
SPT:	1.0814	0.1224	2.2304	1.0020	0.0031	1.0501
BENDER02:	3.6420	2.9046	26.4899	1.1998	0.2230	4.0204
FCFS-DIV:	6.9444	7.7630	65.0442	1.3486	0.4774	7.8634
MCT:	32.4503	10.3601	84.0270	48.3618	15.8681	84.5814
RAND:	6.6204	8.3570	133.7370	1.2166	0.2826	4.9971

A.1.3 Platforms with 20 clusters

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0002	1.0138	1.7129	0.2372	2.9472
ONLINE:	1.0050	0.0321	3.9853	1.0584	0.0401	1.2823
ONLINE-EGDF:	1.0424	0.0647	1.6915	1.0026	0.0032	1.0496
SWRPT:	1.0624	0.0932	1.8335	1.0001	0.0003	1.0092
SRPT:	1.0729	0.0991	1.7652	1.0048	0.0035	1.0255
SPT:	1.0876	0.1231	2.1120	1.0018	0.0028	1.0575
BENDER02:	3.7774	3.0213	23.1174	1.1974	0.2113	4.9324
FCFS-DIV:	6.8593	7.5936	63.0236	1.3292	0.4381	10.4772
MCT:	56.0138	18.5169	144.1156	91.9157	32.1370	161.9829
RAND:	6.8776	8.8223	94.4395	1.2016	0.2515	5.9159

A.2 Number of distinct databases

A.2.1 Platforms with 3 databases

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0002	1.0141	1.4677	0.3132	2.5464
ONLINE:	1.0029	0.0127	1.3012	1.0559	0.0441	1.4158
ONLINE-EGDF:	1.0225	0.0511	1.7488	1.0024	0.0050	1.0935
SWRPT:	1.0329	0.0711	1.8607	1.0002	0.0012	1.0426
SRPT:	1.0384	0.0781	1.7566	1.0035	0.0049	1.1115
SPT:	1.0456	0.0973	2.3820	1.0016	0.0045	1.1814
BENDER98:	1.0275	0.0776	1.8799	1.0023	0.0076	1.1701
BENDER02:	2.3000	2.0686	26.4899	1.1424	0.2468	7.1756
FCFS-DIV:	3.5064	4.8832	58.4923	1.2411	0.5462	18.2120
MCT:	31.2451	19.5496	144.1156	54.6849	36.6557	161.9829
RAND:	3.8050	5.6166	90.8444	1.1695	0.3372	10.3411

A.2.2 Platforms with 10 databases

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0001	1.0071	1.6036	0.2817	2.9472
ONLINE:	1.0044	0.0148	1.1998	1.0493	0.0396	1.3390
ONLINE-EGDF:	1.0426	0.0712	1.6915	1.0027	0.0041	1.0600
SWRPT:	1.0566	0.0913	1.8640	1.0002	0.0009	1.0182
SRPT:	1.0754	0.1077	1.9319	1.0053	0.0049	1.0476
SPT:	1.0860	0.1280	2.2304	1.0023	0.0036	1.0537
BENDER98:	1.0758	0.1265	1.9866	1.0042	0.0065	1.0591
BENDER02:	4.0399	3.1009	25.4483	1.2306	0.2463	4.4978
FCFS-DIV:	7.8969	8.2566	65.0442	1.4290	0.5754	8.9419
MCT:	34.1196	21.3731	134.7519	51.6103	37.0315	156.1615
RAND:	7.4619	9.1068	108.2149	1.2626	0.3656	8.8585

A.2.3 Platforms with 20 databases

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0001	1.0074	1.5796	0.2672	2.7573
ONLINE:	1.0051	0.0328	3.9853	1.0433	0.0340	1.2488
ONLINE-EGDF:	1.0508	0.0762	1.7772	1.0025	0.0035	1.0394
SWRPT:	1.0650	0.0958	1.9632	1.0002	0.0008	1.0183
SRPT:	1.0927	0.1165	1.9075	1.0058	0.0050	1.0485
SPT:	1.1019	0.1335	2.1120	1.0026	0.0035	1.0359
BENDER98:	1.0969	0.1439	2.4877	1.0047	0.0060	1.0399
BENDER02:	4.6555	3.3236	23.6092	1.2492	0.2533	4.9373
FCFS-DIV:	9.8384	9.2215	78.0723	1.4866	0.6212	12.3190
MCT:	36.9529	22.5209	119.9420	51.7317	36.9415	158.3175
RAND:	8.9688	10.3148	133.7370	1.2917	0.4004	8.9272

A.3 Availability of databases

A.3.1 Database probability of existence on a given site : 30%

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0002	1.0141	1.4703	0.2775	2.5464
ONLINE:	1.0047	0.0160	1.3095	1.0530	0.0458	1.4158
ONLINE-EGDF:	1.0471	0.0800	1.7248	1.0035	0.0049	1.0804
SWRPT:	1.0511	0.0896	1.8640	1.0004	0.0011	1.0327
SRPT:	1.0627	0.1014	1.8393	1.0048	0.0050	1.1115
SPT:	1.0626	0.1094	2.3820	1.0010	0.0028	1.0814
BENDER98:	1.0654	0.1258	2.1281	1.0039	0.0074	1.0868
BENDER02:	2.3291	1.9570	22.9606	1.1281	0.1706	4.4978
FCFS-DIV:	4.0977	5.5896	78.0723	1.2235	0.4190	8.9419
MCT:	23.9806	12.3453	89.7986	33.1679	19.1335	112.7939
RAND:	3.9224	5.8244	110.6094	1.1535	0.2989	8.9272

A.3.2 Database probability of existence on a given site : 60%

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0000	1.0000	1.6007	0.2610	2.5336
ONLINE:	1.0051	0.0328	3.9853	1.0486	0.0367	1.3019
ONLINE-EGDF:	1.0505	0.0754	1.7772	1.0028	0.0047	1.0935
SWRPT:	1.0639	0.0962	1.8898	1.0003	0.0011	1.0426
SRPT:	1.0799	0.1100	1.9075	1.0051	0.0050	1.0696
SPT:	1.0860	0.1270	2.1050	1.0018	0.0034	1.1814
BENDER98:	1.0803	0.1346	2.4877	1.0039	0.0076	1.1701
BENDER02:	3.3056	2.4023	25.4483	1.1865	0.1913	3.4303
FCFS-DIV:	5.9105	6.3989	60.7870	1.3132	0.4350	8.7054
MCT:	35.7327	19.8489	123.7441	51.8688	30.4313	130.4241
RAND:	5.4348	6.6610	102.6913	1.1948	0.2669	6.7886

A.3.3 Database probability of existence on a given site : 90%

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0001	1.0071	1.5799	0.3232	2.9472
ONLINE:	1.0027	0.0113	1.3012	1.0468	0.0357	1.2488
ONLINE-EGDF:	1.0184	0.0344	1.4777	1.0012	0.0025	1.0395
SWRPT:	1.0395	0.0745	1.9632	1.0001	0.0005	1.0152
SRPT:	1.0640	0.1013	1.9319	1.0047	0.0049	1.0496
SPT:	1.0849	0.1300	2.2304	1.0037	0.0048	1.0790
BENDER98:	1.0545	0.1047	1.9443	1.0033	0.0052	1.0691
BENDER02:	5.3607	3.6792	26.4899	1.3076	0.3314	7.1756
FCFS-DIV:	11.2335	9.9195	65.0442	1.6200	0.7727	18.2120
MCT:	42.6042	25.2723	144.1156	72.9901	44.7169	161.9829
RAND:	10.8784	11.3979	133.7370	1.3756	0.4769	10.3411

A.4 Workload density

A.4.1 Workload density : 0.75

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0000	1.0000	1.4654	0.2881	2.5336
ONLINE:	1.0013	0.0079	1.2057	1.0198	0.0147	1.1480
ONLINE-EGDF:	1.0247	0.0519	1.5012	1.0008	0.0021	1.0394
SWRPT:	1.0255	0.0548	1.6027	1.0002	0.0008	1.0255
SRPT:	1.0475	0.0932	1.7643	1.0023	0.0037	1.0472
SPT:	1.0392	0.0818	1.9365	1.0010	0.0026	1.0587
BENDER98:	1.0368	0.0896	1.9365	1.0016	0.0045	1.0525
BENDER02:	2.7145	2.3926	22.5747	1.0853	0.1131	3.0139
FCFS-DIV:	4.6039	6.1862	62.2448	1.1508	0.2684	5.9253
MCT:	39.1553	23.6937	144.1156	53.3850	36.9288	154.3724
RAND:	3.7086	5.0954	60.0587	1.0919	0.1657	3.2309

A.4.2 Workload density : 1.00

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0000	1.0000	1.5234	0.2858	2.5426
ONLINE:	1.0019	0.0087	1.1440	1.0274	0.0169	1.1884
ONLINE-EGDF:	1.0271	0.0556	1.5563	1.0012	0.0027	1.0445
SWRPT:	1.0288	0.0578	1.5671	1.0002	0.0009	1.0327
SRPT:	1.0513	0.0924	1.7708	1.0032	0.0041	1.0485
SPT:	1.0480	0.0894	2.0199	1.0014	0.0029	1.0456
BENDER98:	1.0447	0.0933	1.7561	1.0020	0.0048	1.0868
BENDER02:	3.0927	2.6444	22.8899	1.1177	0.1337	2.7041
FCFS-DIV:	5.5388	6.8298	71.9520	1.2137	0.3254	6.0325
MCT:	36.7420	22.3685	134.3679	53.1007	36.8163	152.9597
RAND:	4.7113	6.2268	81.8277	1.1345	0.2309	8.8585

A.4.3 Workload density : 1.25

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0000	1.0000	1.5516	0.2853	2.9472
ONLINE:	1.0026	0.0106	1.1729	1.0355	0.0205	1.2104
ONLINE-EGDF:	1.0308	0.0597	1.6414	1.0016	0.0030	1.0551
SWRPT:	1.0365	0.0662	1.5101	1.0003	0.0012	1.0426
SRPT:	1.0575	0.0955	1.9319	1.0039	0.0046	1.1115
SPT:	1.0564	0.0966	1.8204	1.0017	0.0030	1.0426
BENDER98:	1.0554	0.1159	2.4877	1.0029	0.0070	1.1701
BENDER02:	3.4449	2.8426	22.9606	1.1532	0.1560	2.9629
FCFS-DIV:	6.5364	7.7728	65.0442	1.2777	0.3804	7.2526
MCT:	34.8658	21.3744	127.5231	52.7995	36.8593	159.6311
RAND:	5.6220	7.2981	85.1995	1.1703	0.2481	5.2819

A.4.4 Workload density : 1.50

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0001	1.0074	1.5728	0.2878	2.7333
ONLINE:	1.0032	0.0117	1.1761	1.0440	0.0227	1.1874
ONLINE-EGDF:	1.0354	0.0620	1.6724	1.0021	0.0037	1.0935
SWRPT:	1.0447	0.0735	1.6301	1.0003	0.0010	1.0297
SRPT:	1.0629	0.0934	1.7531	1.0045	0.0041	1.0444
SPT:	1.0711	0.1092	1.9319	1.0022	0.0043	1.1814
BENDER98:	1.0660	0.1185	1.9866	1.0036	0.0062	1.0935
BENDER02:	3.6751	2.9187	20.3285	1.1841	0.1771	2.9620
FCFS-DIV:	7.1922	7.9417	63.1781	1.3362	0.4199	6.2480
MCT:	33.5215	20.5764	106.5829	52.7373	36.9175	154.1226
RAND:	6.3964	7.6214	73.6742	1.2100	0.2784	7.3297

A.4.5 Workload density : 2.00

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0002	1.0141	1.5886	0.2907	2.5459
ONLINE:	1.0053	0.0169	1.3012	1.0626	0.0294	1.2213
ONLINE-EGDF:	1.0444	0.0689	1.7248	1.0030	0.0039	1.0543
SWRPT:	1.0629	0.0919	1.8040	1.0003	0.0009	1.0212
SRPT:	1.0779	0.1046	1.6729	1.0061	0.0047	1.0494
SPT:	1.0946	0.1260	2.0122	1.0026	0.0039	1.0612
BENDER98:	1.0785	0.1262	2.1281	1.0048	0.0071	1.0783
BENDER02:	4.1787	3.2459	23.1174	1.2574	0.2350	4.2284
FCFS-DIV:	8.3660	8.6189	78.0723	1.4770	0.5581	6.8518
MCT:	31.5851	19.6851	106.2579	52.2391	36.7108	152.0034
RAND:	8.1857	9.3891	102.6913	1.2993	0.3466	5.3855

A.4.6 Workload density : 3.00

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
OFFLINE:	1.0000	0.0002	1.0138	1.5998	0.3051	2.7573
ONLINE:	1.0107	0.0468	3.9853	1.1077	0.0451	1.4158
ONLINE-EGDF:	1.0695	0.0920	1.7772	1.0063	0.0063	1.0932
SWRPT:	1.1108	0.1288	1.9632	1.0003	0.0009	1.0147
SRPT:	1.1161	0.1280	1.9075	1.0092	0.0054	1.0696
SPT:	1.1577	0.1713	2.3820	1.0040	0.0053	1.0814
BENDER98:	1.1191	0.1612	2.1814	1.0073	0.0088	1.0730
BENDER02:	4.8847	3.6020	26.4899	1.4466	0.3960	7.1756
FCFS-DIV:	10.2462	9.6467	64.9648	1.8582	0.9777	18.2120
MCT:	28.7654	18.0957	94.9790	51.7919	37.1771	161.9829
RAND:	11.8473	12.7634	133.7370	1.5416	0.5989	10.3411

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

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

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