Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

Minimizing the stretch when scheduling flows of biological requests

Arnaud Legrand ,
Alan Su ,
Frédéric Vivien

October 2005

Research Report N° 2005-48

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

Minimizing the stretch when scheduling flows of biological requests

Arnaud Legrand , Alan Su , Frédéric Vivien

October 2005

Abstract

In this paper, we consider the problem of scheduling comparisons of motifs against biological databanks. This problem lies in the divisible load framework with negligible communication costs. Thus far, very few results have been proposed in this model. We first explain the relationship between this model and the preemptive uni-processor one. After having selected a few relevant metrics (max-stretch and sum-stretch), we show how to extend algorithms that have been proposed in the literature for the uni-processor model to our setting. Then we extensively study the performance of these algorithms in realistic scenarios. Our study clearly suggest an efficient heuristic for each of the two metrics, though a combined optimization is in theory not possible in the general case.

Keywords: Bioinformatics, heterogeneous computing, scheduling, divisible load, linear programming, stretch

Résumé

Nous nous sommes intéressés au problème de l'ordonnancement de requêtes de comparaison de motifs et de bases de données biologiques. Ce problème peut être traiter comme un problème de tâches divisibles dont les temps de communications sont négligeables. Jusqu'à présent, très peu de résultats ont été proposés pour ce modèle. Nous commençons par expliquer les relations entre ce modèle et le modèle mono-processeur avec préemption. Après avoir sélectionné quelques métriques appropriées (max-*stretch* et somme-*stretch*), nous montrons comment étendre les algorithmes de la littérature du cadre mono-processeur à notre cadre de travail. Finalement, nous étudions de manière extensive les propriétés de ces algorithmes dans des scénarios réalistes. Notre étude indique clairement une heuristique efficace pour chacune des deux métriques, bien qu'une optimisation conjointe ne soit en théorie pas possible 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 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 data 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. 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 scheduling problem, motivated specifically by the aforementioned divisible workload scenario and the on-line aspect of the scheduling (i.e., where no knowledge of the whole instance is supposed and where schedulers discover a job's characteristics at its release date). Thus far, no result has been proposed to this specific problem.

Aside from divisibility, the main difference with classical scheduling problems lies in the fact that the platforms we target are shared by many users. In this context, we need to ensure a certain degree of fairness between the different 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 and justified our models 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, and we present known results on closely related problems in Section 4. We also explain how to extend these techniques to our framework and point out the main difficulties. Then, we present in Section 5 an experimental evaluation of the aforementioned techniques. Finally, we conclude and summarize our contributions in Section 6.

2 Applications and Modeling

The GriPPS [5, 7] protein comparison application serves as the context for the scheduling results presented in this paper. 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 homologous 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. From this 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 significant and serve as user input to the application;
3. **sequence comparison servers**: processes co-located with protein databanks, capable of accepting a set of motifs and identifying matches over any subset of the databank.

The following sections describe the scheduling model we use for this application. Experimental validation of this model can be found in [11].

2.1 Fundamental properties: Divisibility, Preemption and Uniform Computation

First, a motif being a compact representation of an amino acid pattern, the communication overhead is negligible compared to the processing time of a comparison. Second, if a pattern is matched against only a subset of databank, then the processing time of such a request is linear with the size of the targeted protein sequence set. This allows us to split 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 to transfer the pattern to the new location, which is negligible).

Note that, 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.

Last, 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. Essentially, $k_{i,i'}$ describes the relative power of processors i and i', regardless of the size or the nature of the job being considered. Our experiments (see [11]) indicate 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, it may be the case in practice that a given databank is not available on all sequence comparison servers. Our setting is then essentially a *uniform machines with restricted availabilities* scheduling problem, which is a specific instance of the more general *unrelated machines* scheduling problem.

2.2 Platform and application model

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 jobs are numbered by increasing release dates. Each job is assigned a *weight* or *priority* w_j. In this article, we focus on the particular case where $w_j = 1/W_j$ but point out a few situations where the general case with arbitrary weights can be solved as well. $p_{i,j}$ 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 requires a databank that is not present on the machine M_i. The time at which job J_j finishes is denoted as C_j. Finally, the *flow time* of the job J_j is defined as $\mathcal{F}_j = C_j - r_j$.

As we have seen in Section 2.1, we could replace the unrelated times $p_{i,j}$ by the expression $W_j \cdot p_i$, where W_j denotes the size (in Mflop) of the job J_j and p_i denotes the computational capacity of machine M_i (in second·Mflop^{-1}). To maintain correctness, we separately track the databanks present at each machine and enforce the constraint that a job J_j may only be executed on a machine at which all dependent data of J_j are present. However, since the work we present does 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 that $p_{i,j}$ may be infinite if J_j cannot be processed on M_i).

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 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: $\forall j, \sum_i \alpha_{i,j} = 1$.

Aside from divisibility, the main difference between this model and classical scheduling problems lies in the fact 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

Let us first note that many interesting results can be found in the scheduling literature. Unfortunately, they only hold for the preemption model (denoted *pmtn*). Thus, in this section, we explain how to extend these techniques to our original problem.

We first recall several common objective functions in the scheduling literature and highlight those that are most relevant in our context. Then, we give a few structural remarks explaining how heuristics for the uni-processor case can be reasonably extended to the general case in our framework, and why the optimization of certain objectives may be mutually exclusive.

3.1 Defining 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. However, individual users 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, $\sum_j \mathcal{F}_j$, suffers from the limitation that starvation is possible, i.e., some jobs may be delayed to an unbounded extent [2]. By contrast, minimization of the maximum flow time, $\max_j \mathcal{F}_j$, 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 [6] focuses on the *weighted* flow time, using job weights to offset the bias against short jobs. *Sum weighted*

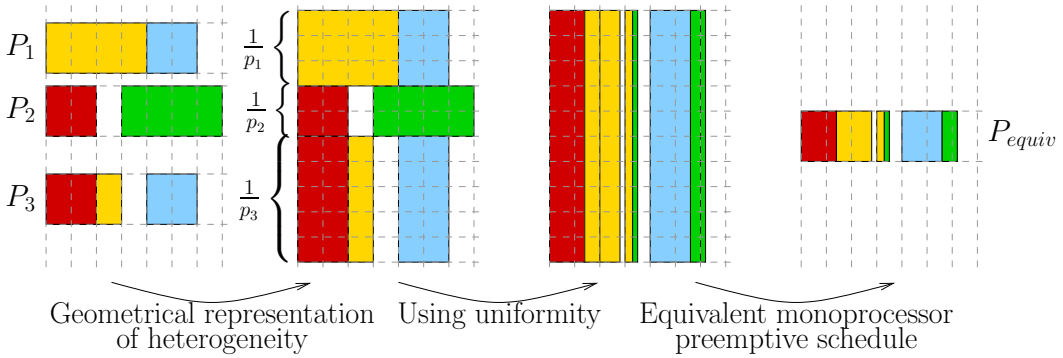


Figure 1: Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem

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 weight is inversely proportional to its size: $w_j = 1/W_j$. The stretch of a job can be seen as the slowdown it experiences when the system is loaded. This is thus a reasonably fair measure of the level of service provided to an individual job and is more relevant than the flow in a system with highly variable job sizes. Consequently, this article focuses mainly on the sum stretch ($\sum \mathcal{S}_j$) and the maximum stretch ($\max \mathcal{S}_j$) metrics.

3.2 A few structural remarks

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.

Lemma 1. *For any instance of n jobs J_1, ..., J_n such that $p_{i,j} = W_j \cdot p_i$, there is a corresponding instance $J_1^{(1)}$, ..., $J_n^{(1)}$ such that:*

For any divisible schedule of J_1, ..., J_n there exists a preemptive schedule of $J_1^{(1)}$, ..., $J_n^{(1)}$ on one processor 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 $1/\sum_i \frac{1}{p_i}$. 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 $p_{i,j} = W_j \cdot p_i$, we define $J_1^{(1)}$, ..., $J_n^{(1)}$ with the same release date as the initial jobs and a processing time $p_j^{(1)} = W_j/(\sum_i \frac{1}{p_i})$. Let us denote by $s^{(t)}$ the starting time of the tth 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 tth and the $(t+1)^{\text{th}}$ 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 $\sum_j \alpha_{i,j}^{(t)} W_j p_i \leqslant \Delta^{(T)}$, hence $\sum_j \alpha_{i,j}^{(t)} W_j \leqslant \frac{\Delta^{(T)}}{p_i}$. Therefore, we have $\sum_i \sum_j \alpha_{i,j}^{(t)} W_j \leqslant \Delta^{(T)} \sum_i \frac{1}{p_i}$ and $\sum_j \left(\sum_i \alpha_{i,j}^{(t)}\right) \frac{W_j}{\sum_i \frac{1}{p_i}} = \sum_j \left(\sum_i \alpha_{i,j}^{(t)}\right) p_j^{(1)} \leqslant \Delta^{(T)}$.

It is thus possible to process $(\sum_i \alpha_{i,j}^{(t)})$ of job $J_j^{(1)}$ 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, completion times can only be decreased. ■

The main idea is that our m heterogeneous processors can be seen as an equivalent processor of power $1/\sum_i \frac{1}{p_i}$. Figure 1 illustrates this idea. The reverse transformation simply processes jobs sequentially, distributing each job's work across all processors. As a consequence, any complexity result for the preemptive uni-processor model also holds for the uniform divisible model. Thus, we study exclusively the uni-processor case in Section 4.

Unfortunately, this line of reasoning does not extend to the restricted availability situation. In the uni-processor case, a schedule can be seen as a priority list of the jobs (see [4] for example). For this reason, the heuristics presented in Section 4 adhere to the same basic principle: maintain a priority list of the jobs and always schedule 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. 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. When restricted availability is introduced

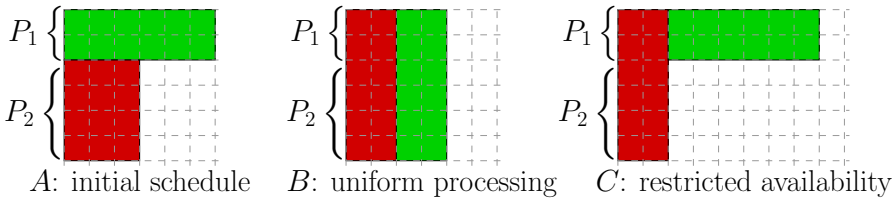


Figure 2: Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem

(situation C), the completion times vectors can no longer be compared, and deciding between one distribution or another is non-trivial in the general case. In such cases, we apply the following simple rule to build a schedule for general platforms from uni-processor heuristics:

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

Last, we want to point out that the criteria we have defined earlier may be in opposition.

Theorem 1. *Let Δ be the ratio of the sizes of the largest and shortest jobs submitted to the system. Consider any on-line 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 $\rho(\Delta) < \Delta$. Then, there exists for this algorithm a sequence of jobs leading to starvation and for which the obtained max-stretch is arbitrarily greater than the optimal max-stretch.*

We can also show that for an on-line algorithm which has a competitive ratio of $\rho(\Delta)$ 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, under the constraints that $\rho(\Delta) < \frac{1+\Delta}{2}$.

Proof. First, we must comment on our assumption about *non-trivial competitive ratios*. If all jobs have the same size, then FIFO is obviously an optimal schedule for sum-stretch, and thus has a competitive ratio of 1. Therefore, if we ignore the size of jobs and schedule FIFO all jobs considering them as unit-size jobs, and if we multiply by Δ all the dates defined by the schedule obtained this way, we would end up with a trivial schedule of competitive ratio at most Δ.

We first consider the case of an on-line algorithm for the sum-stretch optimization problem, whose competitive ratio is $\rho(\Delta)$. At date 0 arrives a job J_1 of size Δ. Let k be any integer. Then, at any time unit $t \in \mathbb{N}$, $t \leqslant k-1$, starting from time 0, arrives a job J_{1+t} 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 job J_1. The sum-stretch is then $\left(1+\frac{k}{\Delta}\right)+k$ and the max-stretch is $1+\frac{k}{\Delta}$.

In fact, with our hypotheses, the on-line algorithm cannot complete the execution of the job J_1 as long as there are jobs of size 1 arriving at each time unit. Otherwise, suppose that at the date t_1, job J_1 was completed. Then, a certain number k_1 of unit-size jobs were completed before time t_1. The scenario which minimizes the sum-stretch is to schedule the first k_1 jobs at their release date, then to schedule J_1, and then the remaining $k-k_1$ jobs. The sum-stretch of the actual schedule can therefore not be smaller than the sum-stretch of this schedule, which is equal to: $\left(1+\frac{k_1}{\Delta}\right)+k_1+(k-k_1)(1+\Delta)$. 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:

$$\left(1+\frac{k_1}{\Delta}\right)+k_1+(k-k_1)(1+\Delta) \leqslant \rho(\Delta)\left(1+\frac{k}{\Delta}+k\right) \quad \Leftrightarrow$$
$$(1-\rho(\Delta))+k_1\left(\frac{1}{\Delta}-\Delta\right) \leqslant k\,(1+\Delta)\left(\frac{\rho(\Delta)}{\Delta}-1\right).$$

Once the approximation algorithm has completed the execution of the job J_1, 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 $\frac{\rho(\Delta)}{\Delta}-1 \geqslant 0$, i.e., $\rho(\Delta) \geqslant \Delta$, which contradicts our hypothesis on the competitive ratio.

Therefore, the only possible behavior for the approximation algorithm is to delay the execution of job J_1 until after the end of the arrival of the unit-size jobs, whatever the number of these jobs. This leads to starvation of job J_1. Furthermore, the ratio of the obtained max-stretch to the optimal one is $\frac{1+\frac{k}{\Delta}}{1+\Delta} = \frac{\Delta+k}{\Delta(\Delta+1)}$, 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 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 Heuristics for the uni-processor case

In this section, we consider only scheduling problems on one processor, due to the demonstration in the previous section of the equivalence of the "uniform machines with divisibility" and "uni-processor with preemption" models. We first recall the known results for flow minimization. Then we move to sum- and, finally, max-stretch minimization.

4.1 Minimizing max- and sum-flow

Flow minimization is generally a simple problem when considering on-line preemptive scheduling on a single processor. Indeed, the *first come, first served* heuristic optimally minimizes the max-flow [2]. Also, the *shortest remaining processing time* first heuristic (SRPT) minimizes the sum-flow [1].

4.2 Minimizing the sum-stretch

The complexity of the off-line minimization of the sum-stretch is still an open problem. At the very least, this is a hint at the difficulty of this problem.

Bender, Muthukrishnan, and Rajaraman presented in [4] a Polynomial Time Approximation Scheme (PTAS) for minimizing the sum-stretch. In [6] Chekuri and Khanna presented an approximation scheme for the more general sum weighted flow minimization problem. As these approximation schemes cannot be extended to work in an on-line setting, we will not discuss them further.

In [13], Muthukrishnan, Rajaraman, Shaheen, and Gehrke prove that there is no optimal on-line algorithm for the sum-stretch minimization problem when there are three or more distinct job sizes. They also give a lower bound of 1.036 on the competitive ratio of any on-line algorithm. They propose an optimal on-line algorithm when there are only two job sizes.

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 takes into account 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 [13] that SRPT is 2-competitive for sum-stretch.

Another well studied algorithm is the Smith's ratio rule 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 [14] 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, we have no result on the efficiency of SWPT for the minimization of the sum stretch. Note that 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 weakness of SWPT heuristics is obviously that it does not take into account the remaining processing times: it may preempt a job at the moment 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 time of the jobs. This is what the *shortest weighted remaining processing time* heuristic (SWRPT) does. At any time t, SWRPT schedules the job J_j that minimizes $\frac{\rho_t(j)}{W_j}$. Therefore, in the framework of sum-stretch minimization, at any time t, SWRPT schedules the job J_j which minimizes $\frac{1}{p_j \rho_t(j)}$.

Neither of the proofs of competitiveness of SRPT or SWPT can be extended to SWRPT. SWRPT has apparently been studied by N. Megow in [12], but only in the scope of the sum weighted completion time. So far, there exists no guarantee on the efficiency of SWRPT. 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 2. *For any real $\varepsilon > 0$, there exists an instance such that* SWRPT *is not $(2-\varepsilon)$-competitive for the minimization of the sum stretch.*

Proof. The whole proof can be found in the Appendix (Section A). Here we just present the construction used to prove this lower bound on the competitive ratio of SWRPT.

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.

Formally, we build the instance $\mathcal{J}$ as follows (n, k and l will be defined below):

1. Job J_0 arrives at time $r_0 = 0$ and is of size 2^{2^n}.
2. Job J_1 arrives at time $r_1 = 2^{2^n} - 2^{2^{n-2}}$ and is of size $2^{2^{n-1}}$.
3. Job J_2 arrives at time $r_2 = r_1 + 2^{2^{n-1}} - \alpha$ and is of size $2^{2^{n-2}}$, where $\alpha = 1 - \frac{\varepsilon}{3}$.
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 $2^{2^{n-j}}$.
5. Job J_{n+j}, for $1 \leqslant j \leqslant k$, is of size $2^{2^{-j}}$ 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 1 and arrives at time $r_{n+k+j} = r_{n+k+j-1} + p_{n+k+j-1}$.

where $n = \left\lceil \log_2 \left(\log_2 \frac{3(1+\alpha)}{\varepsilon} \right) \right\rceil$ and $k = \lceil -\log_2(-\log_2 \alpha) \rceil$. Then, we evaluate the sum-stretch realized by both SWRPT and SRPT and we show that, if l is large enough, the sum-stretch realized by SWRPT is $(\mathcal{R} \geqslant 2-\varepsilon)$-times that realized by SRPT. This proves the result as the optimal sum-stretch is no greater than that of SRPT. ■

4.3 Minimizing the maximum stretch

4.3.1 The off-line case

We have previously shown [11, 10] that the maximum weighted flow – a generalization of the maximum stretch – can be minimized in polynomial time when the release dates and jobs are known in advance (i.e., in the off-line framework). This problem can in fact be solved for a set of unrelated processors, and we briefly describe our solution in its full generality below. For a more detailed presentation, readers are referred to the above-mentioned publications.

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 $\mathcal{F}$. Then, for each job J_j, we define a deadline $\bar{d}_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ (to minimize the maximum stretch, just let $w_j = 1/p_j$). Then, the maximum weighted flow is no greater than the objective $\mathcal{F}$, if and only if the execution of each job J_j is completed before its deadline. 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.

Let us suppose that 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 $]\mathcal{F}_1, \mathcal{F}_2[$, 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 not a constant but an affine function in $\mathcal{F}$. 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 then affine functions in $\mathcal{F}$. Using these definitions and notations, System (1) searches the objective interval $[\mathcal{F}_1, \mathcal{F}_2]$ for the minimal maximum weighted flow achievable.

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

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 at at most one point). Let n_{q} be the number of distinct milestones. Then, $1 \leqslant n_{\mathrm{q}} \leqslant n^2 - n$. We denote by $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_{\mathrm{q}}}$ the milestones ordered by increasing values. To solve our problem we just need to perform a binary search on the set of milestones $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_{\mathrm{q}}}$, each time checking whether System (1) has a solution in the objective interval $[\mathcal{F}_i, \mathcal{F}_{i+1}]$ (except for $i = n_{\mathrm{q}}$ in which case we search for a solution in the range $[\mathcal{F}_{n_{\mathrm{q}}}, +\infty[)$. This process can be performed in its entirety in polynomial time, as the linear programs have rational variables.

4.3.2 The on-line case

Bender, Chahrabarti, and Muthukrishnan have shown in [2] that the minimization of the max-stretch in an on-line setting is a very difficult problem. They indeed proved the following lower bound on the competitive ratio of any on-line algorithm:

Theorem 3. *For three lengths of jobs there is no $\frac{\Delta^{\frac{1}{3}}}{2}$-competitive on-line algorithm for max-stretch.*[2]

We first recall two existing on-line algorithms before introducing a new one. In [3], Bender, Muthukrishnan, and Rajaraman 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 on-line algorithm.

Bender, Chahrabarti, and Muthukrishnan had already described in [2] another $O(\sqrt{\Delta})$-competitive on-line 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 (off-line) 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 to $\alpha = \sqrt{\Delta}$.

This last on-line algorithm has several weaknesses. The first is that, when they designed their algorithm, Bender et al. did not know how to compute the (off-line) optimal maximum stretch. This is now overcome. Another weakness in this approach is that such an algorithm tries only to optimize the stretch of the most constraining jobs. In other words, 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 5. This problem could be ameliorated by specifying that each job should be scheduled in a manner that minimizes its 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 (2).

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

$$\text{UNDER THE CONSTRAINTS} \quad \begin{cases} \text{(2a)} & \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t(\mathcal{S}^*) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(2b)} & \forall i, \forall j, \forall t, \quad \bar{d}_j(\mathcal{S}^*) \leqslant \inf I_t(\mathcal{S}^*) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(2c)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)} . p_{i,j} \leqslant \sup I_t(\mathcal{S}^*) - \inf I_t(\mathcal{S}^*) \\ \text{(2d)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \end{cases} \tag{2}$$

This system ensures that each job is completed no later than the deadline defined by the optimal (off-line) max-stretch. Then, under this constraint, this system attempts to minimize an objective that resembles a rational relaxation of the

[1] In [8], Labetoulle, Lawler, Lenstra, and Rinnoy Kan call such a value a "critical trial value".
[2] Here Δ is once again the ratio of the largest to the smallest job size.

sum-stretch: it computes the sum of the average execution times of the different jobs. 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. We further refine this heuristic to define the following on-line 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 (2).

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 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 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 next section. Note that in Step (2), we look for the best achievable max-stretch knowing what scheduling decisions were already taken, i.e., knowing which jobs were already (partially) executed, and when, whereas Bender et al. were looking for the optimal max-stretch.

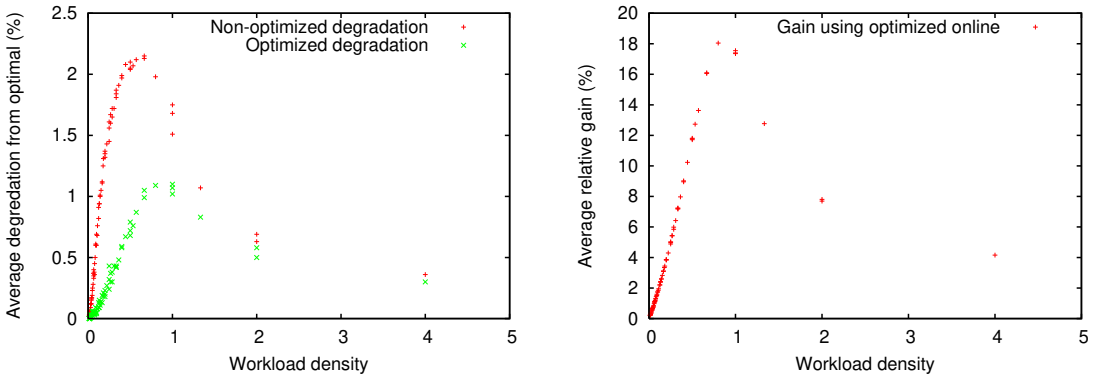
5 Simulation results

In order to evaluate the efficacy of various scheduling strategies when optimizing stretch-based metrics, we implement a simulation framework using the SimGrid toolkit [9], 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 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.

5.1 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.

1. **Platform size:** Typically, a given biological databases 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 clusters in the simulated platform. Without loss of generality, we arbitrarily define each site to contain 10 processors.
2. **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.



(a) Max-stretch degradation from optimal of both versions of the on-line heuristic

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

Figure 3: Comparison of the optimized and non-optimized versions of the on-line heuristic

3. **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.

4. **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.

5. **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.

6. **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 handle 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 clusters 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 generate realistic values for the database sizes and the processor speeds. The remaining four parameters – platform size, number of distinct databases, database availability, and workload density – represent the experimental values for our study. We discuss further the specifics of the experimental design and our simulation results in Section 5.3.

5.2 Optimization of the on-line heuristic

In order to motivate the variants of our on-line heuristic described in Section 4.3.2, we conduct a series of experiments to evaluate their effect. In particular, we consider a non-optimized version of the on-line 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 3 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 on-line heuristic. The max-stretch of each is then divided by the max-stretch

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0003	1.0167	1.6729	0.3825	4.4468
Online	1.0025	0.0127	2.0388	1.0806	0.0724	2.0343
Online-EDF	1.0024	0.0127	2.0581	1.0775	0.0708	2.0392
Online-EGDF	1.0781	0.1174	2.4053	1.0021	0.0040	1.0707
Bender98 [3]	1.0798	0.1315	2.0978	1.0024	0.0044	1.0530
SWRPT	1.0845	0.1235	2.5307	1.0002	0.0012	1.0458
SRPT	1.0939	0.1299	2.3741	1.0044	0.0055	1.0907
SPT	1.1147	0.1603	2.8295	1.0027	0.0054	1.1195
Bender02	3.4603	3.0260	28.4016	1.2053	0.2417	5.2022
MCT-Div	6.3385	7.4375	73.4019	1.3732	0.5628	11.0440
MCT	27.0124	20.1083	129.6119	50.9840	36.9797	157.8909

Table 1: Aggregate statistics over all 162 platform/application configurations

achieved by the optimal algorithm, yielding a degradation factor for both heuristics on that run. However, since the optimal sum-stretch is not known, we observe the sum-stretch of the optimized on-line heuristic relative to the non-optimized version. Figures 3(a) and 3(b) present the max-stretch and sum-stretch results, respectively. In the first graph plots average max-stretch degradation values compared to the optimal result against workload density, for each of 80 configurations; each data point represents the average degradation over 5000 simulations. The second plot depicts the average improvement in sum-stretch (for the same 80 configurations and 5000 runs per configuration) obtained when using the optimized heuristic, relative to the non-optimized version. These results strongly motivate the use of the the optimizations encoded by the linear program depicted in System (2).

5.3 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 4.3.1 that solves the optimal max-stretch problem. The three versions of the on-line heuristic are also implemented, designated as Online, Online-EDF, and Online-EGDF. Next, we consider the SWRPT, SRPT, and SPT heuristics discussed in Section 4. 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 MCT-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 3). Note that neither of these two heuristics make any changes to work that has already been scheduled. Finally, we consider the two on-line heuristics proposed by Bender et al. that were briefly described in Section 4.3.2.

As mentioned earlier, two of the six parameters of our model reflect empirical values determined in our previous work with the GriPPS system [11]. 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 above-mentioned heuristics as we vary four parameters:

- **platforms** of 3, 10, and 20 clusters with 10 processors each;
- **applications** with 3, 10, and 20 distinct reference 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 1 presents the aggregate results from these simulations; finer-grained results based on various partitionings of the data may be found in the Appendix (Section B).

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 27 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: MCT-Div is on average 6.3 times worse

in terms of max-stretch than the best approach we found. One of the principal failings of the MCT and MCT-Div heuristics is that they are non-preemptive. As the stretch of small tasks are fairly sensitive to wait time, such a task arriving at a moment when the system is loaded will be stretched substantially.

Experimentally, we find that two of the three on-line heuristics we propose are consistently near-optimal (within 0.1% on average) for max-stretch optimization. The third, Online-EGDF actually achieves consistently good sum-stretch, but at the expense of its performance for the max-stretch metric. 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 1% of optimal for all configurations. In particular, SWRPT produces a sum-stretch that is on average 0.02% 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 nth job). Our analysis of the GriPPS application logs has revealed that such situations occur quite often due to some automatic submission processes. By optimizing max-stretch in lieu of sum-stretch, the possibility of starvation is eliminated.

Next, we find that the Bender98 and Bender02 heuristics are not practically useful in our scheduling context. The results shown in Table 1 for the Bender98 heuristic comprise only 3-cluster platforms; simulations on larger platforms were practically infeasible, due to the algorithm's prohibitive overhead costs. Effectively, for an n-task workload, the Bender98 heuristic solves n optimal max-stretch problems, many of which are computationally equivalent to the full n-task optimal solution. In several cases the desired workload density required thousands of tasks, rendering the Bender98 algorithm intractable. To roughly compare the overhead costs of the various heuristics, we ran a small series of simulations using only 3-cluster platforms. The results of these tests indicate that the scheduling time for a 15-minute workload was on average under 0.28 s for any of our on-line heuristics, and 0.54 s for the off-line optimal algorithm (with 0.35 s spent in the resolution of the linear program and 0.19 s spent in the on-line 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 the competitive ratios it guarantees are ineffective compared with our on-line heuristics for max-stretch optimization.

Finally, we note the anomalous result that optimal algorithm is occasionally beaten (in all cases by a variant of the on-line heuristic); clearly this indicates an error in the solution of the optimal max-stretch problem. Our preliminary 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. This may result in a very small time interval (ranging between two nearly identical values of $\mathcal{F}$), which then might be missed by the search process because it fails to distinguish the $\mathcal{F}$ values. 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.

6 Conclusion

Our main contributions to this problem are the following:

- We present a synthesis of existing theoretical work on the closely related uni-processor model with preemption and explain how to extend most results to our particular setting.
- Although this idea was underlying in previous work (e.g., in [2]), we prove the impossibility of simultaneously approximating both max-based and sum-based metrics.
- We note that the natural heuristic for sum-stretch optimization, SWRPT, is not well-studied. Although this simple heuristic seems to optimize the sum-stretch in practice, its performance is not guaranteed. However, we prove that it cannot be guaranteed with a factor better than 2.
- We propose an on-line strategy for max-stretch optimizations in this problem domain, and we demonstrate its efficacy using a wide range of realistic simulation scenarios. All previously proposed guaranteed heuristics for max-stretch (Bender98 and Bender02) for the uni-processor model prove to be particularly inefficient in practice.
- On average, our various on-line algorithms based on linear programs prove to be near-optimal solutions for max-stretch. SRPT and SWRPT, which were originally designed to optimize the sum-stretch metric, surprisingly yield fairly good results for the max-stretch metric. However, due to the potential for starvation with sum-based metrics, we assert that our max-stretch optimization heuristics are preferable for job- and user-centric systems.

[3] Bender98 results are limited to 3-cluster platforms, due to prohibitive overhead costs (discussed in Section 5.3).

References

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

[2] Michael A. Bender, Soumen Chahrabarti, 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. ACM press, 1998.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

A Lower bound on the competitive ratio of SWRPT

Proof. The 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 size one. Each job arrives at a date equal to the release date of its predecessor plus the execution time of its predecessor, except for the second and third jobs which arrives at dates critical for SWRPT.

Formally, we build the instance $\mathcal{J}$ as follows. Let $\alpha = 1 - \frac{\varepsilon}{3}$. Let n k, and l be three integers such that : $n = \left\lceil \log_2 \log_2 \frac{3(1+\alpha)}{\varepsilon} \right\rceil$, and $k = \lceil -\log_2(-\log_2 \alpha) \rceil$.

1. Job J_0 arrives at time $r_0 = 0$ and is of size 2^{2^n}.
2. Job J_1 arrives at time $r_1 = 2^{2^n} - 2^{2^{n-2}}$ and is of size $2^{2^{n-1}}$.
3. Job J_2 arrives at time $r_2 = r_1 + 2^{2^{n-1}} - \alpha$ and is of size $2^{2^{n-2}}$.
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 $2^{2^{n-j}}$.
5. Job J_{n+j}, for $1 \leqslant j \leqslant k$, is of size $2^{2^{-j}}$ 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 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.

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

  $\rho_{r_2}(J_2) = 2^{2^{n-2}}$. 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:

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

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) 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:

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

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

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

We then chose 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,

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

Therefore, we can chose l large enough to have

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

Then,

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

■

B Additional simulation results

Tables 2 through 4 present the simulation data for 3-, 10-, and 20-site configurations. Tables 5 through 10 present the simulation data for configurations with specific workload densities, varying from 0.75 to 3.00. Tables 11 through 13 present the simulation data for 3-, 10-, and 20-database configurations. Tables 14 through 16 present the simulation data for configurations with database availability of 30%, 60%, and 90%.

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0001	1.0057	1.4346	0.3406	3.2160
Online	1.0012	0.0083	1.2648	1.0604	0.0557	1.7044
Online-EDF	1.0011	0.0082	1.2648	1.0548	0.0530	1.7017
Online-EGDF	1.0557	0.1027	2.0936	1.0017	0.0037	1.0566
SWRPT	1.0643	0.1153	2.5307	1.0002	0.0013	1.0433
SRPT	1.0728	0.1205	2.1328	1.0042	0.0061	1.0907
SPT	1.0949	0.1595	2.8295	1.0033	0.0063	1.1195
Bender02	3.1209	2.8235	28.4016	1.2178	0.2922	5.2022
MCT-Div	6.4998	7.9212	68.3501	1.4771	0.7660	11.0440
MCT	10.3419	4.0266	121.6338	16.7938	4.8924	46.8819

Table 2: Aggregate statistics over 54 platform/application configurations using 3 sites

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0003	1.0167	1.7582	0.3548	3.9253
Online	1.0026	0.0113	1.2634	1.0950	0.0832	2.0343
Online-EDF	1.0025	0.0112	1.2634	1.0923	0.0808	2.0392
Online-EGDF	1.0838	0.1223	2.1460	1.0022	0.0037	1.0707
SWRPT	1.0884	0.1247	2.1469	1.0002	0.0010	1.0251
SRPT	1.0971	0.1306	2.1469	1.0044	0.0045	1.0333
SPT	1.1182	0.1582	2.3381	1.0025	0.0043	1.0448
Bender02	3.4492	2.9337	27.5690	1.1993	0.2178	3.5167
MCT-Div	6.3270	7.4253	73.4019	1.3367	0.4500	7.3333
MCT	25.0726	12.1027	83.1075	46.3988	16.8691	84.9341

Table 3: Aggregate statistics over 54 platform/application configurations using 10 sites

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0004	1.0165	1.8255	0.3313	4.4468
Online	1.0037	0.0169	2.0388	1.0865	0.0711	1.9958
Online-EDF	1.0037	0.0171	2.0581	1.0853	0.0699	1.9863
Online-EGDF	1.0949	0.1225	2.4053	1.0024	0.0046	1.0588
SWRPT	1.1006	0.1275	2.0754	1.0001	0.0011	1.0458
SRPT	1.1117	0.1351	2.3741	1.0047	0.0059	1.0333
SPT	1.1311	0.1609	2.4130	1.0022	0.0053	1.0625
Bender02	3.8102	3.2639	27.3621	1.1990	0.2056	3.5672
MCT-Div	6.1890	6.9315	54.1129	1.3060	0.3802	5.6269
MCT	45.5868	20.5669	129.6119	89.6846	33.2259	157.8909

Table 4: Aggregate statistics over 54 platform/application configurations using 20 sites

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0003	1.0148	1.6636	0.4310	4.4468
Online	1.0008	0.0057	1.1244	1.0420	0.0443	1.9958
Online-EDF	1.0008	0.0057	1.1244	1.0388	0.0394	1.7131
Online-EGDF	1.0392	0.0715	1.6490	1.0007	0.0025	1.0477
SWRPT	1.0413	0.0737	1.6490	1.0001	0.0010	1.0215
SRPT	1.0528	0.0908	1.9064	1.0021	0.0044	1.0616
SPT	1.0591	0.1033	1.9130	1.0012	0.0037	1.0796
Bender02	2.6110	2.4933	27.3621	1.0886	0.1196	2.6219
MCT-Div	4.2758	5.8801	57.8379	1.1587	0.2978	7.1549
MCT	30.7513	22.6511	129.6119	51.6552	37.0841	154.5800

Table 5: Aggregate statistics over 27 platform/application configurations with workload density of 0.75

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0002	1.0087	1.6815	0.4013	3.6012
Online	1.0011	0.0068	1.1765	1.0546	0.0511	1.6325
Online-EDF	1.0010	0.0066	1.1765	1.0505	0.0463	1.5247
Online-EGDF	1.0493	0.0817	1.8226	1.0009	0.0026	1.0490
SWRPT	1.0523	0.0850	1.8226	1.0001	0.0009	1.0205
SRPT	1.0650	0.1027	1.8226	1.0027	0.0046	1.0521
SPT	1.0746	0.1185	2.0091	1.0016	0.0044	1.1001
Bender02	2.9802	2.7600	28.4016	1.1175	0.1321	3.0905
MCT-Div	5.1722	6.6865	68.3501	1.2093	0.3189	6.0890
MCT	29.0574	21.1960	118.9077	51.5397	36.9930	152.1818

Table 6: Aggregate statistics over 27 platform/application configurations with workload density of 1.00

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0004	1.0165	1.6873	0.3835	3.9253
Online	1.0017	0.0086	1.1490	1.0670	0.0553	1.7945
Online-EDF	1.0016	0.0086	1.1556	1.0615	0.0508	1.7877
Online-EGDF	1.0623	0.0936	1.7260	1.0013	0.0030	1.0311
SWRPT	1.0671	0.0987	1.7649	1.0001	0.0009	1.0226
SRPT	1.0779	0.1118	2.1469	1.0035	0.0051	1.0907
SPT	1.0933	0.1323	2.0929	1.0022	0.0047	1.0957
Bender02	3.2584	2.8377	26.5854	1.1506	0.1511	2.4128
MCT-Div	5.8173	6.8755	60.7281	1.2690	0.3637	5.8874
MCT	27.7061	20.1537	107.3472	51.2116	36.9157	157.8909

Table 7: Aggregate statistics over 27 platform/application configurations with workload density of 1.25

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0004	1.0167	1.6898	0.3734	3.2586
Online	1.0020	0.0102	1.2634	1.0744	0.0575	1.7630
Online-EDF	1.0020	0.0102	1.2634	1.0734	0.0571	1.7352
Online-EGDF	1.0739	0.1039	1.7812	1.0017	0.0035	1.0707
SWRPT	1.0786	0.1077	1.9008	1.0002	0.0013	1.0433
SRPT	1.0899	0.1195	1.9914	1.0041	0.0051	1.0440
SPT	1.1079	0.1445	2.4130	1.0025	0.0049	1.0583
Bender02	3.4825	2.9844	25.9149	1.1826	0.1767	3.1846
MCT-Div	6.3037	7.1902	60.4304	1.3240	0.4200	6.2201
MCT	26.4973	19.5775	94.3396	50.7819	36.8234	157.7347

Table 8: Aggregate statistics over 27 platform/application configurations with workload density of 1.50

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0002	1.0084	1.6801	0.3566	3.3490
Online	1.0030	0.0118	1.2390	1.0995	0.0721	1.8607
Online-EDF	1.0030	0.0117	1.2390	1.0979	0.0716	1.8497
Online-EGDF	1.1006	0.1269	2.0188	1.0026	0.0040	1.0476
SWRPT	1.1069	0.1312	1.9647	1.0002	0.0012	1.0277
SRPT	1.1159	0.1379	1.9647	1.0056	0.0054	1.0373
SPT	1.1430	0.1668	2.6495	1.0034	0.0059	1.1195
Bender02	3.9233	3.2009	27.5690	1.2574	0.2295	4.0166
MCT-Div	7.4813	7.9766	55.3821	1.4696	0.5681	9.4111
MCT	24.9462	18.5232	95.2381	50.4874	36.8712	156.0182

Table 9: Aggregate statistics over 27 platform/application configurations with workload density of 2.00

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0002	1.0070	1.6349	0.3399	2.9322
Online	1.0063	0.0236	2.0388	1.1461	0.0909	2.0343
Online-EDF	1.0063	0.0237	2.0581	1.1427	0.0905	2.0392
Online-EGDF	1.1433	0.1669	2.4053	1.0054	0.0056	1.0588
SWRPT	1.1601	0.1754	2.5307	1.0003	0.0016	1.0458
SRPT	1.1614	0.1695	2.3741	1.0087	0.0058	1.0561
SPT	1.2102	0.2190	2.8295	1.0051	0.0071	1.1148
Bender02	4.5031	3.4066	23.2689	1.4347	0.3627	5.2022
MCT-Div	8.9719	8.7093	73.4019	1.8075	0.8904	11.0440
MCT	23.1295	17.1353	121.6338	50.2310	37.1835	156.9455

Table 10: Aggregate statistics over 27 platform/application configurations with workload density of 3.00

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0003	1.0167	1.4979	0.3444	3.3299
Online	1.0024	0.0113	1.3026	1.0701	0.0564	1.7044
Online-EDF	1.0024	0.0111	1.3026	1.0655	0.0539	1.7017
Online-EGDF	1.0592	0.1095	2.1947	1.0022	0.0047	1.0707
SWRPT	1.0639	0.1174	2.5307	1.0003	0.0018	1.0458
SRPT	1.0690	0.1185	2.1328	1.0035	0.0055	1.0907
SPT	1.0808	0.1497	2.8295	1.0021	0.0061	1.1195
Bender02	2.3317	2.0982	22.4182	1.1401	0.2223	5.2022
MCT-Div	3.2875	4.5014	62.0873	1.2246	0.4815	11.0440
MCT	27.0797	18.8117	129.6119	53.5436	36.7236	157.8909

Table 11: Aggregate statistics over 54 platform/application configurations with 3 reference databases

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0003	1.0166	1.7476	0.3742	4.4468
Online	1.0027	0.0153	2.0388	1.0870	0.0821	2.0343
Online-EDF	1.0026	0.0154	2.0581	1.0845	0.0807	2.0392
Online-EGDF	1.0854	0.1192	2.0460	1.0021	0.0038	1.0561
SWRPT	1.0924	0.1263	2.0659	1.0001	0.0007	1.0205
SRPT	1.1020	0.1314	2.1469	1.0048	0.0056	1.0565
SPT	1.1255	0.1625	2.4009	1.0029	0.0051	1.0796
Bender02	3.8022	3.1393	28.4016	1.2306	0.2509	4.3492
MCT-Div	7.1260	7.5863	68.3501	1.4255	0.5959	10.1591
MCT	26.5667	20.2844	117.3514	49.7426	37.0234	157.7347

Table 12: Aggregate statistics over 54 platform/application configurations with 10 reference databases

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0003	1.0165	1.7732	0.3662	4.1263
Online	1.0023	0.0111	1.2634	1.0848	0.0751	1.9958
Online-EDF	1.0024	0.0112	1.2634	1.0825	0.0734	1.8497
Online-EGDF	1.0897	0.1208	2.4053	1.0020	0.0035	1.0323
SWRPT	1.0971	0.1240	2.1458	1.0001	0.0005	1.0133
SRPT	1.1106	0.1354	2.3741	1.0050	0.0055	1.0411
SPT	1.1379	0.1626	2.6495	1.0031	0.0049	1.0462
Bender02	4.2474	3.3475	27.5690	1.2453	0.2374	3.8653
MCT-Div	8.6029	8.5496	73.4019	1.4696	0.5736	9.4838
MCT	27.3910	21.1527	111.3333	49.6653	37.0615	149.3393

Table 13: Aggregate statistics over 54 platform/application configurations with 20 reference databases

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0001	1.0041	1.6418	0.4515	4.4468
Online	1.0016	0.0096	1.1991	1.1178	0.0968	2.0343
Online-EDF	1.0015	0.0094	1.1765	1.1115	0.0957	2.0392
Online-EGDF	1.0742	0.1203	2.4053	1.0024	0.0038	1.0588
SWRPT	1.0690	0.1154	2.3263	1.0003	0.0015	1.0458
SRPT	1.0706	0.1126	2.1328	1.0041	0.0046	1.0565
SPT	1.0883	0.1461	2.6785	1.0018	0.0044	1.0864
Bender02	2.0534	1.9157	28.4016	1.1277	0.1771	4.3492
MCT-Div	3.6172	5.4143	68.3501	1.2344	0.4738	10.3450
MCT	14.5871	8.7936	121.6338	30.5590	18.2418	115.3582

Table 14: Aggregate statistics over 54 platform/application configurations with database availability of 30%

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0003	1.0167	1.7546	0.3262	3.7500
Online	1.0028	0.0151	2.0388	1.0726	0.0507	1.7044
Online-EDF	1.0028	0.0153	2.0581	1.0705	0.0494	1.7017
Online-EGDF	1.0960	0.1267	2.0936	1.0025	0.0043	1.0561
SWRPT	1.1025	0.1352	2.0936	1.0002	0.0012	1.0373
SRPT	1.1083	0.1364	2.0912	1.0047	0.0055	1.0561
SPT	1.1266	0.1657	2.8295	1.0024	0.0048	1.1148
Bender02	2.9329	2.0364	27.5690	1.1826	0.1834	4.0166
MCT-Div	4.9589	5.2580	73.4019	1.2980	0.4053	8.0257
MCT	27.0743	16.7717	91.4105	50.1104	30.3253	128.8167

Table 15: Aggregate statistics over 54 platform/application configurations with database availability of 60%

	Max-stretch			Sum-stretch		
	Mean	SD	Max	Mean	SD	Max
Offline	1.0000	0.0004	1.0165	1.6222	0.3442	3.2160
Online	1.0031	0.0128	1.2715	1.0515	0.0386	1.3593
Online-EDF	1.0030	0.0127	1.2715	1.0504	0.0384	1.3593
Online-EGDF	1.0642	0.1014	2.1947	1.0013	0.0039	1.0707
SWRPT	1.0818	0.1166	2.5307	1.0000	0.0006	1.0240
SRPT	1.1027	0.1359	2.3741	1.0045	0.0064	1.0907
SPT	1.1294	0.1649	2.5322	1.0039	0.0065	1.1195
Bender02	5.3951	3.6954	27.3621	1.3057	0.3060	5.2022
MCT-Div	10.4401	9.1034	67.1243	1.5873	0.7005	11.0440
MCT	39.3782	23.3925	129.6119	72.2866	44.4828	157.8909

Table 16: Aggregate statistics over 54 platform/application configurations with database availability of 90%