Minimizing the Stretch When Scheduling Flows of Biological Requests

Arnaud Legrand
Laboratoire ID-IMAG, France
Arnaud.Legrand@imag.fr

Alan Su
Google Inc., Cambridge, MA, USA
su.alan@gmail.com

Frédéric Vivien
INRIA - LIP, ENS Lyon, France
Frederic.Vivien@inria.fr

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, derive new lower bounds on the competitive ratio of any on-line algorithm, present new competitiveness results for existing algorithms, and develop several new online heuristics. 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 particularly 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.

Categories and Subject Descriptors

F.2.2 [**Analysis of algorithms and problem complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

General Terms

Algorithms, Theory

Keywords

Scheduling, stretch, flow time, divisible load, online algorithm, competitive analysis, linear programming.

SPAA'06, July 30–August 2, 2006, Cambridge, Massachusetts, USA.

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 model 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 the location of replicas 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 application model, particularly in the context of *on-line 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 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. In Section 4, we recall known results on closely related problems, derive new lower bounds on the competitive ratio of any on-line algorithm, present new competitiveness results for existing algorithms, and develop several new online heuristics. Then, we present in Section 5 an experimental evaluation of the aforementioned heuristics. Finally, we conclude and summarize our contributions in Section 6. An extended version of this article with fully elaborated proofs and analyses is available as a companion research report [13].

2. APPLICATIONS AND MODELING

The GriPPS [6, 8] 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 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. 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 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 following sections describe the scheduling model we use for this application domain, which we have validated in our previous work [12].

2.1 Model properties: divisibility, preemption and uniform computations

First, we note that a motif is a relatively compact representation of an amino acid pattern, and thus the communication overhead is negligible compared to the processing time of a comparison. Second, 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 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). 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 [12]) 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 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 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 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 $\mathcal{F}_j = C_j - r_j$, 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 requires a databank that is not present on the machine M_i. 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}). We denote by Δ the ratio of the sizes of the largest and shortest jobs submitted to the system: $\Delta = \frac{\max_j W_j}{\min_j W_j}$.

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).

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 we also point out a few conditions under which the general case with arbitrary weights can be solved as well.

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

An 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

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

We first recall several common objective functions in the scheduling literature and highlight those that are most relevant for our problem. 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 [3]. 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 [7] 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: $w_j = 1/W_j$. 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 max-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 platform M_1, ..., M_m composed of uniform processors, i.e., such that for any job J_j, $p_{i,j} = W_j \cdot p_i$, one can define a platform made of a single processor $\widetilde{M}$ with $\widetilde{p} = 1/\sum_i \frac{1}{p_i}$, 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.

Figure 1 illustrates the underlying idea (see [11] for details). The reverse transformation simply processes jobs sequentially, distributing each job's work across all processors. As a consequence, any complexity result for the preemptive uni-processor model also holds for the uniform divisible model. Thus, in Section 4, in addition to addressing the multi-processor case, we will also closely examine the uniprocessor 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 [5] for example). For this reason, the heuristics for the uniprocessor case presented in Section 4 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. 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:

```
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.
```

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

THEOREM 1. *Consider any on-line algorithm that 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 that leads to starvation, and thus for which the obtained max-stretch is arbitrarily greater than the optimal max-stretch.*

We can also show that for an on-line algorithm that 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 $\rho(\Delta) < \frac{1+\Delta}{2}$. (Remember that Δ is the ratio of the sizes of the largest and shortest jobs submitted to the system.)

We must comment on our assumption about *non-trivial competitive ratios*. This comes from the fact that ignoring job sizes leads to a Δ-competitive on-line algorithm for both objective functions:

THEOREM 2. First come, first served *is Δ-competitive for the on-line minimization of sum-stretch and for the on-line minimization of max-stretch.*

We now prove Theorem 1.

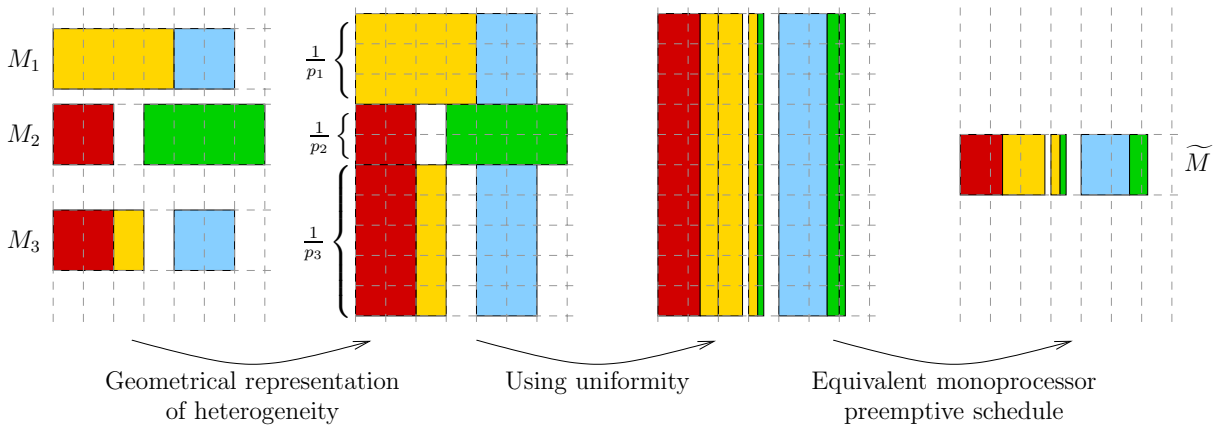


Figure 1: Geometrical transformation of a divisible uniform problem into a preemptive uni-processor problem.

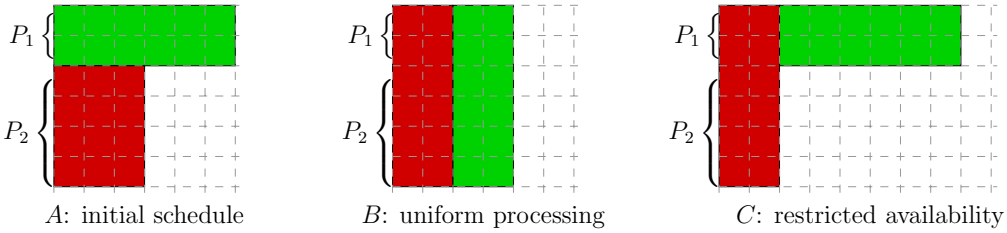


Figure 2: Illustrating the difference between the uniform model and the restricted availability model.

PROOF. We first consider the case of an on-line algorithm for the sum-stretch optimization problem that achieves a competitive ratio of $\rho(\Delta)$. 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 that 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:

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

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 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. ON-LINE ALGORITHMS AND HEURISTICS

In this section, we first recall the known results for flow minimization. Then we move to sum- and, finally, max-stretch minimization. All results for sum-flow, max-flow and sum-stretch optimization focus on uni-processor platforms, as we have seen in the previous section the equivalence of the "uniform machines with divisibility" and "uni-processor with preemption" models. Through this equivalence, we can directly reuse these results in our framework.

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 [3]. 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 sum-stretch minimization is still an open problem. At the very least, this hints at the difficulty of this problem.

Bender, Muthukrishnan, and Rajaraman presented in [5] a Polynomial Time Approximation Scheme (PTAS) for minimizing the sum-stretch. In [7] 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 [15], Muthukrishnan, Rajaraman, Shaheen, and Gehrke propose an optimal on-line algorithm when there are only two job sizes. Their primary result shows that there is no optimal on-line 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 on-line algorithm. The following theorem improves this bound:

THEOREM 3. *No on-line algorithm minimizing the sum-stretch has a competitive ratio less than or equal to 1.19484.*

We have recalled that *shortest remaining processing time* (SRPT) is optimal for minimizing the sum-flow. When SRPT makes 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, this implies that SRPT does not account for the *weight* of the jobs in the objective function. Nevertheless, Muthukrishnan, Rajaraman, Shaheen, and Gehrke have shown [15] that SRPT is 2-competitive for sum-stretch minimization.

Another well studied algorithm is Smith's ratio rule [17] also known as *shortest weighted processing time* (SWPT). This is a preemptive list scheduling heuristic that orders available jobs for execution by increasing value of the ratio $\frac{W_j}{w_j}$. Whatever the weights, SWPT is 2-competitive [16] 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 above bound for 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, first note that SWPT schedules the available jobs by increasing values of $(W_j)^2$ (since examining job stretch implies that $w_j = \frac{1}{W_j}$) and has thus exactly the same behavior as the *shortest processing time* heuristic (SPT). Then the following theorem states that SPT – and thus SWPT – is not an approximation algorithm for minimizing the sum-stretch.

THEOREM 4. *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.*

The weakness of the SWPT heuristic as an on-line sum-stretch minimization strategy is not surprising, since it does not take into account the work that has already been done on jobs in the system. Due to this fact, it may preempt a job just before the moment when it would finish, ignoring the potentially large stretch penalty such a decision incurs.

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 $\frac{\rho_t(j)}{W_j}$, where $\rho_t(j)$ is the remaining processing time of job J_j at time t. Therefore, in the framework of sum-stretch minimization, at any time t, SWRPT schedules the job J_j which minimizes $p_j \rho_t(j)$. Neither of the proofs of competitiveness of SRPT or SWPT can be extended to SWRPT. SWRPT has apparently been studied by N. Megow in [14], 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 5. *For any real $\varepsilon > 0$, there exists an instance such that* SWRPT *is not $(2-\varepsilon)$-competitive for the minimization of the sum-stretch.*

4.3 Minimizing the max-stretch

4.3.1 The off-line case.

We have previously shown (in [12]) that the maximum weighted flow – a generalization of the max-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 publication.

It should be noted that, prior to our work, a solution to compute the maximum weighted flow on a set of uniform machines, using network flow maximization techniques, was described. See [2] for an example of a pre-existing solution to the max-stretch minimization problem. We do not know how to extend these flow maximization techniques to handle the uniform machines with restricted availabilities case, much less the more general case of unrelated processors.

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 $w_j = 1/W_j$). 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 that 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 that 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 (where $\alpha_{i,j}^{(t)}$ denotes the fraction of job J_j processed on M_i during time interval I_t).

$$\begin{aligned}
&\text{MINIMIZE} \quad \mathcal{F} \\
&\text{UNDER THE CONSTRAINTS} \\
&\begin{cases}
\mathcal{F}_1 \leqslant \mathcal{F} \leqslant \mathcal{F}_2 \\
\forall i, \forall j, \forall t,\ r_j \geqslant \sup I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\forall i, \forall j, \forall t,\ \bar{d}_j(\mathcal{F}) \leqslant \inf I_t(\mathcal{F}) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\forall t, \forall i,\ \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leqslant \sup I_t(\mathcal{F}) - \inf I_t(\mathcal{F}) \\
\forall j,\ \sum_t \sum_i \alpha_{i,j}^{(t)} = 1
\end{cases}
\end{aligned} \tag{1}$$

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, and at most $\frac{n(n-1)}{2}$ milestones at which two distinct deadline functions coincides. Let n_{q} be the number of distinct milestones. Then, $1 \leqslant n_{\text{q}} \leqslant n^2 - n$. Let $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_{\text{q}}}$ be the milestones ordered by increasing values. To solve our problem we perform a binary search on the set of milestones, each time checking whether System (1) has a solution in the objective interval $[\mathcal{F}_i, \mathcal{F}_{i+1}]$ (except for $i = n_{\text{q}}$, in which case we search for a solution in the range $[\mathcal{F}_{n_{\text{q}}}, +\infty[$). 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.

The minimization of the max-stretch in an on-line setting is a very difficult problem. First, *first come, first served* (FCFS), which was optimal for minimizing max-flow, is only Δ-competitive for max-stretch (cf. Theorem 2). In contrast, consider the fact that the optimal algorithm for sum-flow, SRPT, is 2-competitive for sum-stretch. The seemingly poor performance of FCFS is better understood in light of the following lower bound on the competitive ratio of *any* on-line algorithm:

THEOREM 6. *Given a workload composed of jobs of three distinct sizes, there is no $\frac{1}{2}\Delta^{\sqrt{2}-1}$-competitive preemptive on-line algorithm (for a uni-processor machine) for max-stretch minimization.*

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

[1] In [9], Labetoulle, Lawler, Lenstra, and Rinnooy Kan call such a value a "critical trial value".

As $O(\sqrt{\Delta})$-competitive algorithms are known to exist, this result has bridged roughly half of the gap between the previous lower bound and the best algorithms from the literature.

We first recall two existing on-line algorithms before introducing a new one. In [4], Bender, Muthukrishnan, and Rajaraman defined, for any job J_j, a pseudo-stretch $\widehat{\mathcal{S}}_j(t)$:

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

Then, jobs were scheduled in order of 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, Chakrabarti, and Muthukrishnan had already described in [3] 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: $\bar{d}_j(\mathcal{F}) = r_j + \lambda \times \mathcal{S}^*/W_j$, where λ is a constant termed the *expansion factor*. Finally, a schedule is realized by executing jobs according to their deadlines, using the *Earliest Deadline First* strategy. To achieve the optimal competitive ratio, Bender et al. set $\lambda = \sqrt{\Delta}$.

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 it optimizes the stretch of only 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 approach could be ameliorated by specifying 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, to derive a new on-line heuristic for multi-processor platforms, we consider the heuristic approach expressed by System (2).

$$\begin{aligned}
&\text{MINIMIZE} \quad \sum_{j=1}^{n} \sum_{t} \left(\sum_{i=1}^{m} \alpha_{i,j}^{(t)} \right) \frac{\sup I_t(\mathcal{S}^*) + \inf I_t(\mathcal{S}^*)}{2} \\
&\text{UNDER THE CONSTRAINTS} \\
&\begin{cases}
\forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t(\mathcal{S}^*) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\forall i, \forall j, \forall t, \quad \bar{d}_j(\mathcal{S}^*) \leqslant \inf I_t(\mathcal{S}^*) \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.p_{i,j} \leqslant \sup I_t(\mathcal{S}^*) - \inf I_t(\mathcal{S}^*) \\
\forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1
\end{cases}
\end{aligned} \tag{2}$$

This system ensures that each job is completed no later than the deadline defined by the optimal (off-line) max-stretch, $\mathcal{S}^*$. Then, under this constraint, this system attempts to minimize an objective that resembles a rational relaxation of the sum-stretch: it computes an approximation of the sum of the average execution time of the each 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. Consequently, this heuristic offers no guarantee on the sum-stretch achieved. Conceptually, the on-line refinement of our approach follows this basic heuristic:

- Each time a new job arrives:
  1. Preempt the running jobs (if any).
  2. Compute the best achievable max-stretch $\mathcal{S}^*$, considering the amount of each job's work already completed.
  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 are scheduled to complete during that 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 using a list scheduling approach where 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 accounting for the dynamic status of jobs, i.e., knowing for each job how much work remains at the instant the scheduling decision is made. By contrast, Bender et al. considered only the optimal max-stretch of the whole instance, ignoring the amount of work already completed.

5. SIMULATION RESULTS

In order to evaluate the efficacy of various scheduling strategies when optimizing stretch-based metrics, we implemented a simulator using the SimGrid toolkit [10], 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.

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 exists 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 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 ran-

dom 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 experimental values that vary in 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

We conduct a preliminary series of experiments to evaluate the variants of our on-line heuristic from the previous section. These comparisons are based on a non-optimized version of the on-line heuristic that stops after Step 2 of the strategy presented. For a range of simulation configurations, we record the max- and sum-stretch of jobs in the workload achieved with both methods. The max-stretch of each is measured relative to the optimal algorithm, but 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. Figure 3(a) presents the max-stretch degradation of both versions relative to the optimal, and Figure 3(b) 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 (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 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 3.2). Note that neither MCT nor FCFS-Div makes 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. 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 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 [12]. 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 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.

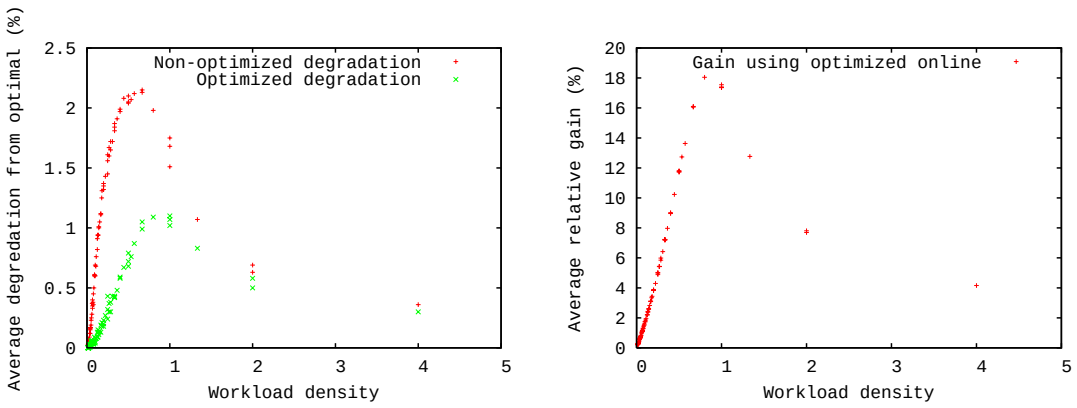
The resulting experimental framework has 162 configurations. For each configuration, 200 platforms and application instances are randomly generated and the simulation results for each of the studied heuristics is recorded. Table 1 presents the aggregate results from these simulations; finer-grained results based on various partitionings of the data may be found in [11].

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: FCFS-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 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 two of the three on-line heuristics we propose are consistently near-optimal (within 0.3‰ on average) for max-stretch optimization. The third heuristic, 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 4‰ 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 n^{th} job). Our

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



(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.

	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 [2]	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
FCFS-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.

analysis of the GriPPS application logs has revealed that such situations occur fairly often due to automated processes that submit jobs at regular intervals. By optimizing max-stretch in lieu of sum-stretch, the possibility of starvation is eliminated.

Next, we find that the BENDER98 and BENDER02 heuristics are not practically useful in our scheduling context. The results shown in Table 1 for the BENDER98 heuristic comprise only 3-cluster platforms; simulations on larger platforms were practically infeasible, due to the algorithm's prohibitive overhead costs. Effectively, for an n-task workload, the BENDER98 heuristic solves n optimal max-stretch problems, many of which are computationally equivalent to the full n-task optimal solution. In several cases the desired workload density required thousands of tasks, rendering the BENDER98 algorithm intractable. To roughly compare the overhead costs of the various heuristics, we ran a small series of simulations using only 3-cluster platforms. The results of these tests indicate that the scheduling time for a 15-minute workload was on average under 0.28 s for any of our 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 in realistic scenarios for our application domain, the competitive ratios it guarantees are ineffective compared with our on-line heuristics for max-stretch optimization.

Finally, we note the anomalous result that the 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. 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 principal 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 for this model to our particular setting.

- Although this idea was underlying in previous work (e.g., in [3]), we prove the impossibility of simultaneously approximating both max-based and sum-based metrics.
- We improve the existing lower bounds on the competitive ratio of any on-line algorithm for our two metrics of interest: sum-stretch and max-stretch.
- 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. We even prove that it cannot be guaranteed with a factor better than 2.
- Despite the fact that FCFS-Div is the only algorithm that exhibits guaranteed performance (Δ-competitive) for both criteria, we observe that the experimental performance of FCFS-Div is substantially worse than that of other algorithms that take full advantage of preemption.
- We propose an on-line strategy for max-stretch optimization 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 the max-stretch optimization heuristics we develop are preferable for job- and user-centric systems.

7. REFERENCES

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

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

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

[4] M. A. Bender, S. Muthukrishnan, and R. 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.

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

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

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

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

[9] J. Labetoulle, E. L. Lawler, J. Lenstra, and A. 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.

[10] A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Proceedings of the 3rd IEEE Symposium on Cluster Computing and the Grid*, 2003.

[11] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of biological requests. Research report RR2005-48, École Normale Supérieure de Lyon, Oct. 2005.

[12] A. Legrand, A. Su, and F. Vivien. Off-line scheduling of divisible requests on an heterogeneous collection of databanks. In *Proceedings of the 14th Heterogeneous Computing Workshop*, Denver, Colorado, USA, Apr. 2005. IEEE Computer Society Press.

[13] A. Legrand, A. Su, and F. Vivien. Minimizing the stretch when scheduling flows of divisible requests. Research Report RR2006-19, LIP, École Normale Supérieure de Lyon, June 2006.

[14] N. Megow. Performance analysis of on-line algorithms in machine scheduling. Diplomarbeit, Technische Universität Berlin, Apr. 2002.

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

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

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