INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Off-line scheduling of divisible requests on an heterogeneous collection of databanks

Arnaud Legrand — Alan Su — Frédéric Vivien

N° 5386

November 2004

Thème NUM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--5386--FR+ENG

Off-line scheduling of divisible requests on an heterogeneous collection of databanks

Arnaud Legrand, Alan Su, Frédéric Vivien

Thème NUM — Systèmes numériques
Projet GRAAL

Rapport de recherche n° 5386 — November 2004 — 13 pages

Abstract: In this paper, we consider the problem of scheduling comparisons of motifs against biological databanks. We show that this problem lies in the divisible load framework. In this framework, we propose a polynomial-time algorithm to solve the maximum weighted flow off-line scheduling problem on unrelated machines. We also show how to solve the maximum weighted flow off-line scheduling problem with preemption on unrelated machines.

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

This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

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

Ordonnancement de requêtes divisibles sur une collection hétérogène de bases de données

Résumé : Nous nous sommes intéressés au problème de l'ordonnancement de requêtes de comparaison de motifs et de bases de données biologiques. Nous avons montré expérimentalement que ce problème peut être traiter comme un problème de tâches divisibles. Dans ce contexte, nous avons proposé un algorithme en temps polynomial qui produit un ordonnancement minimisant le flot pondéré maximal sur un ensemble de machines de caractéristiques non corrélées, et ce quand les dates d'arrivée des tâches sont connues à l'avance (modèle off-line). Nous montrons également comment construire des ordonnancements minimisant le flot pondéré maximal quand les tâches ne sont pas divisibles mais seulement préemptibles.

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

1 Introduction

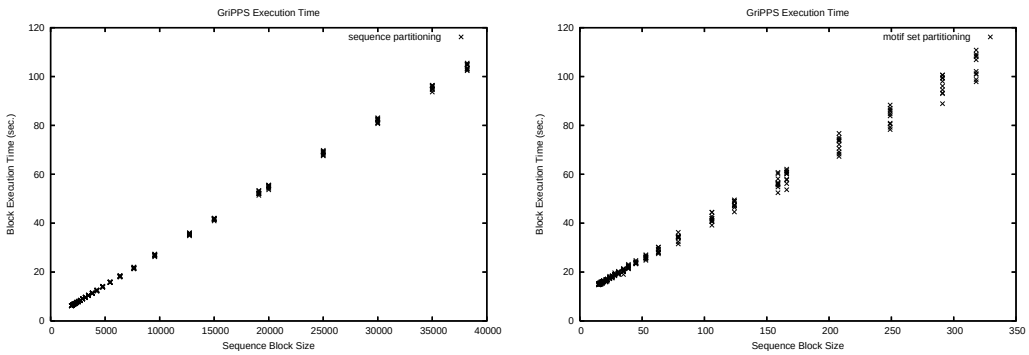
The problem of searching large-scale genomic sequence databases is an increasingly important bioinformatics problem. The results we present in this paper concern the deployment of such applications in heterogeneous parallel computing environments. In fact, this application is a part of a larger class 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. For example, several parallel implementations of the BLAST [1] and FASTA [9] sequence comparison algorithms have been realized for various computational environments (e.g. [3, 4, 8]). 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. The work presented in this paper concerns this scheduling problem, motivated specifically by the aforementioned divisible workload scenario. Our work differs from prior work primarily in the theoretical model we consider, which admits a platform composed of fully unrelated processors. We believe the generality of this approach will enable us to apply our scheduling strategies in a wide range of heterogeneous platforms.

The remainder of this paper is organized as follows. Section 2 introduces the GriPPS protein comparison application, a genomic sequence comparison application as described above. The GriPPS system serves as the archetype for our application and distributed computing platform models, presented in Section 3. Section 4 describes our theoretical results: given a series of comparison tasks and a distributed platform on which they are to be executed, we show a polynomial-time algorithm to identify the optimal value for the *maximum weighted flow* metric and an application schedule that achieves that optimum. We solve this problem both in the divisible load framework and in the more classical framework with task preemption. Finally, we conclude by discussing our planned extensions to this work in Section 5.

2 Framework

The GriPPS protein comparison application serves as the context for the scheduling results presented in this paper. To develop a suitable application model, we performed a series of experiments to analyze the fundamental properties of the sequence comparison algorithms used in this code. The principal components of this application are: 1) **protein databanks** – large 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; and



(a) Sequence databank divisibility

(b) Motif set divisibility

Figure 1: Divisibility studies

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.

We performed an initial set of experiments to demonstrate that the GriPPS application workload exhibits a high degree of *divisibility* – comparisons of a set of motifs against a large sequence database can be partitioned into many independent sub-tasks that have aggregate computational requirements equivalent to that of the original task itself. In these experiments we consider a fixed set of roughly 300 motifs and a database of approximately 38,000 protein sequences. We consider a series of partition sizes for the protein database, ranging from the full sequence set to subsets of roughly 1900 (1/20 of the full set). For each subset size, we perform ten iterations with a subset of that size, with the sequences chosen randomly from the complete set. We then launch a GriPPS search using the full set of motifs and the constructed sequence subset, and we record the total elapsed time for that comparison. Figure 1(a) depicts the measured execution time for these requests, according to the task size. These results indicate that the GriPPS workload is highly divisible, as the correlation between task size and computation time is nearly perfectly linear.

We also ran a second series of experiments to evaluate the impact of inter-processor communication on application performance. We similarly partitioned the set of approximately 300 motifs into subsets of varying size. We then invoked the GriPPS comparison application to find matches between each motif subset among the entire reference sequence database. The results of these experiments are presented in Figure 1(b). Our findings indicate a fundamental difference in the manner in which motifs and sequences are treated by the algorithms used in the GriPPS framework: although computation costs vary roughly linearly with the size of the motif subset chosen, a fixed overhead cost is evident from

the empirical data. To quantify the difference between the observations shown in Figure 1, we performed linear regression analyses on both datasets to project the significance of this fixed overhead. In the motif partitioning experiments, the overhead was estimated to be 10.5 seconds, whereas the overhead for sequence set partitioning was 1.1 seconds.

Finally, we performed a set of experiments to study the time needed to send the full motif set across a typical cluster interconnection network, and the time to report the results of a corresponding GriPPS application invocation over that same network; our results indicate that these communication overhead costs are negligible, compared to the computational workload in typical usage scenarios. Due to these results, we neglect data transfer costs in this paper.

The GriPPS protein databank search application is an example of a *divisible workload* due to the (i) linear relationship between the job computation costs and the size of the targeted protein sequence set, and (ii) the negligible communication overheads. In this paper, we present scheduling strategies that take advantage of these properties. We now present formal models for divisible workloads and the distributed heterogeneous platforms we are targeting.

3 Platform and application model

Notations. Formally, an instance of our problem is defined by n jobs, J_1, ..., J_n and m machines, 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 arrive ordered by increasing release dates. Each job is assigned a *weight* or *priority* w_j. $c_{i,j}$ denotes the amount of time it would take for machine M_i to process job J_j. Note that $c_{i,j}$ can be infinite if the job J_j requires a database that is not present on the machine M_i. The time at which job J_j finishes is denoted as C_j. The *flow* of the job J_j is defined as $F_j = C_j - r_j$.

For the GriPPS application described earlier, all machines process the same type of jobs. In this context, we could replace the unrelated times $c_{i,j}$ by the expression $W_j \cdot c_i$, where W_j denotes the size (in Mflop) of the job J_j and c_i denotes the computational capacity of machine M_i (in second·Mflop^{-1}). To maintain correctness, we separately track the databases 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. Thus, the problem is essentially a *uniform machines with restricted availabilities* scheduling problem, which is a specific instance of the more general *unrelated machines* scheduling problem. However, since the work we present does not rely on these restrictions, we retain the more general (i.e., unrelated machines) scheduling problem formulation.

Job divisibility. 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 each of these machines is working on a different part 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: $\forall j, \sum_i \alpha_{i,j} = 1$. Note that, from a theoretical perspective, divisible load is a generalization of the *preemptive execution model* that allows for simultaneous execution of different parts of a same job on different processors.

Objective function. The most common objective function in the parallel scheduling literature is the *makespan*, i.e., the maximum of the job termination time $\max_j C_j$. Makespan minimization is conceptually a system-centric perspective, seeking to ensure efficient platform utilization. However, individual users are typically more interested in optimizing *job flow* (also called *response time*), i.e., the time their jobs spend in the system. Optimizing the average (or total) flow time, $\sum_j F_j$, suffers from the limitation that starvation is possible, i.e., some jobs may be delayed to an unbounded extent [2]. By contrast, minimization of the maximum flow time, $\max_j F_j$, does not exhibit this drawback, but it tends to favor long jobs to the detriment of short ones. We therefore focus on the maximum *weighted* flow time, using job weights to offset the bias against short jobs. *Maximum stretch* is a particular case of maximum weighted flow, in which a job weight is equal to its size $w_j = W_j$. Bender, Chakrabarti, and Muthukrishnan have shown in [2] that, on a single machine, no polynomial time algorithm can approximate the non-preemptive max-stretch problem to within a factor of $\Omega(n^{1-\epsilon})$ for arbitrarily small $\epsilon > 0$ unless P=NP. Moreover, they state that the preemptive version admits a fully polynomial time approximation scheme (FPTAS). We now show that under our divisible load hypothesis, we are able to solve the maximum weighted flow scheduling problem on unrelated machines in polynomial time.

4 Minimizing the maximum weighted flow

In this theoretical study, we examine the *off-line* version of the problem: we suppose that for each job, the scheduler knows (in advance) its size, its data dependencies, and its release date. In future work, we will use the results of the *off-line* study to propose solutions to the *on-line* problem, in which the scheduler discovers a job's characteristics at its release date.

Section 4.1 describes the solution of the makespan minimization problem in the divisible load framework for our application model. We then discuss in Section 4.2 the problem of deadline scheduling and its polynomial-time solution in the same application context. These results are then extended in Section 4.3, which presents a solution to the minimization of the maximum weighted flow problem in the divisible load framework. By adapting some of these techniques, we then describe a solution to the problem of minimization of the maximum weighted flow when preemption (but not load divisibility) is allowed; these results are given in Section 4.4.

4.1 Makespan minimization

In this section we consider the classical problem of the minimization of the makespan. The release dates sorted by increasing values, r_1, ..., r_n, along with $+\infty$, define a set of n_{int}

time intervals $I_1, \ldots, I_{n_{\text{int}}}$. If all release dates are distinct, then $n_{\text{int}} = n$ and $I_j = [r_j, r_{j+1}[$ (except $I_n = [r_n, +\infty[$). We denote each time interval I_t by $I_t = [\inf I_t, \sup I_t[$. We further define $\alpha_{i,j}^{(t)}$ as the fraction of job J_j processed by processor P_i during the time interval I_t. In this framework, Linear Program (1) lists the constraints that should hold true in any valid schedule:

1. *release date*: job J_j cannot be processed before it is released (Equation (1a));
2. *resource usage*: during a time interval, a processor cannot be used longer than the duration of this time interval (Equation (1b));
3. *end of schedule*: during the last interval, I_n, any processor is used a time at most Δ_n (Equation (1c));
4. *job completion*: each job must be processed to completion (Equation (1d)).

Regarding the objective function of Linear Program (1), we first remark that the processing of the final job J_n cannot start sooner than its release date, r_n. Thus, $C_{\max}$ occurs at a point in time equal to the release date of the final job plus Δ_n the latest processor completion time for the final interval, I_n. Hence, the given objective function represents the makespan.

$$\text{Minimize} \quad C_{\max} = r_n + \Delta_n \quad \text{under the constraints}$$
$$\begin{cases} \text{(1a)} & \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(1b)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leqslant \sup I_t - \inf I_t \\ \text{(1c)} & \forall i, \quad \sum_j \alpha_{i,j}^{(n)}.c_{i,j} \leqslant \Delta_n \\ \text{(1d)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \end{cases} \tag{1}$$

Any feasible solution to Linear Program (1) gives us a straightforward optimal solution to the makespan minimization problem: during any time interval I_t we can schedule in any order (and without idle times) the non-null fractions $\alpha_{i,j}^{(t)}$. Since Linear Program (1) only has rational variables:

Theorem 1. *Minimizing the makespan is a polynomial problem.*

4.2 Deadline scheduling

In the framework of *deadline scheduling*, each job J_j has not only a release date r_j but also a deadline $\bar{d}_j$. The problem is then to find a schedule such that each job J_j is executed within its executable time interval $[r_j, \bar{d}_j]$.

Consider the set of all job release dates and job deadlines: $\{r_1, \ldots, r_n, \bar{d}_1, \ldots, \bar{d}_n\}$. We define an *epochal time* as a time value at which one or more points in this set occurs; there

are between 2 (when all jobs are released at the same date and have the same deadline) and $2n$ (when all job release dates and job deadlines are distinct) such values. When ordered in absolute time, adjacent epochal times define a set of *time intervals*, analogous to the time intervals constructed solely from release dates in the previous section. Let us again denote by $I_1, \ldots, I_{n_{\text{int}}}$ this set of time intervals, noting that $1 \leq n_{\text{int}} \leq 2n - 1$. Accordingly, given an interval I_t, we can reuse the definitions for (i) the interval lower bound ($\inf I_t$), (ii) the interval upper bound ($\sup I_t$), and (iii) the division and assignment of tasks to processors during these intervals ($\alpha_{i,j}^{(t)}$). In this framework, System (2) lists the constraints that should hold true in any valid schedule:

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

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

Lemma 1. *System (2) has a solution if, and only if, there exists a schedule satisfying, for any job J_j, the release date r_j and the deadline $\bar{d}_j$.*

System (2) can be solved in polynomial time by any linear solver system as all our variables are rational. Building a valid schedule from any solution of System (2) is straightforward as in any time interval I_t the job fractions $\alpha_{i,j}^{(t)}$ can be scheduled in any order.

4.3 Minimizing the maximum weighted flow

4.3.1 Relationship with deadline scheduling

Let us assume that we are looking for a schedule $\mathcal{S}$ under which the maximum weighted flow is less than or equal to some objective value $\mathcal{F}$. The weighted flow of any job J_j is equal to $w_j(C_j - r_j)$. Then, the execution of J_j must be terminated before time $\bar{d}_j(\mathcal{F}) = r_j + \mathcal{F}/w_j$ for $\mathcal{S}$ to satisfy the bound $\mathcal{F}$ on the maximum weighted flow. Therefore, looking for a schedule which satisfies a given upper bound on the maximum weighted flow is equivalent to an instance of the deadline scheduling problem.

One may think that by applying a binary search on possible values of the objective value $\mathcal{F}$, one would be able to find the optimal maximum weighted flow, and an optimal schedule.

However, a binary search on this value is not guaranteed to terminate, as it can not attain any arbitrary value of a rational interval. By setting a limit on the precision on the binary search, the number of process iterations is bounded, and the quality of the approximation can be guaranteed. We now show how to adapt our search to always find the optimal in polynomial time.

4.3.2 Problem resolution

So far we have used System (2) to check whether our problem has a solution whose maximum weighted flow is smaller than some objective value $\mathcal{F}$. We now show that we can use it to check whether our problem has a solution for some particular *range* of objective values. Later we show how to divide the whole search space into a number of search ranges polynomial in our problem size.

Solving on a range. First, let us suppose there exist two values $\mathcal{F}_1$ and $\mathcal{F}_2$, $\mathcal{F}_1 < \mathcal{F}_2$, such that the relative order of the release dates and deadlines, $r_1, \ldots, r_n, \bar{d}_1(\mathcal{F}), \ldots, \bar{d}_n(\mathcal{F})$, when ordered in absolute time, is independent of the value of $\mathcal{F} \in]\mathcal{F}_1; \mathcal{F}_2[$. Then, on the objective interval $]\mathcal{F}_1, \mathcal{F}_2[$, as before, we define an epochal time as a time value at which one or more points in the set $\{r_1, \ldots, r_n, \bar{d}_1(\mathcal{F}), \ldots, \bar{d}_n(\mathcal{F})\}$ occurs. Note that an epochal time which corresponds to a deadline is no longer a constant but an affine function in $\mathcal{F}$. As previously, when ordered in absolute time, adjacent epochal times define a set of *time intervals*, that we denote by $I_1, \ldots, I_{n_{\text{int}}(\mathcal{F})}$. The durations of time intervals are now affine functions in $\mathcal{F}$. Using these new definitions and notations, we can solve our problem on the objective interval $[\mathcal{F}_1, \mathcal{F}_2]$ using System (2) with the additional constraint that $\mathcal{F}$ belongs to $[\mathcal{F}_1, \mathcal{F}_2]$ ($\mathcal{F}_1 \leqslant \mathcal{F} \leqslant \mathcal{F}_2$), and with the minimization of $\mathcal{F}$ as the objective. This gives us System (3).

$$\text{Minimize} \quad \mathcal{F} \quad \text{under the constraints}$$
$$\begin{cases} \text{(3a)} & \mathcal{F}_1 \leqslant \mathcal{F} \leqslant \mathcal{F}_2 \\ \text{(3b)} & \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(3c)} & \forall i, \forall j, \forall t, \quad \bar{d}_j \leqslant \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(3d)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)} . c_{i,j} \leqslant \sup I_t - \inf I_t \\ \text{(3e)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \end{cases} \tag{3}$$

Particular objectives. The relative ordering of the release dates and deadlines only changes for values of $\mathcal{F}$ where one deadline coincides with a release date or with another deadline. We call such a value of $\mathcal{F}$ a *milestone*.[1] In our problem, there are at most n distinct

[1] In [6], Labetoulle, Lawler, Lenstra, and Rinnoy Kan call such a value a "critical trial value".

release dates and as many distinct deadlines. Thus, there are at most $\frac{n(n-1)}{2}$ milestones at which a deadline function coincides with a release date. There are also at most $\frac{n(n-1)}{2}$ milestones at which two deadline functions coincides (two affine functions intersect in at most one point). Let n_{q} be the number of distinct milestones. Then, $1 \leqslant n_{\mathrm{q}} \leqslant n^2 - n$. 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 (2) 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 $[\mathcal{F}_{n_{\mathrm{q}}}, +\infty[$). Hence, we have the following theorem:

Theorem 2. *Minimizing the optimal maximum weighted flow is a polynomial problem.*

4.4 Minimizing the maximum weighted flow with preemption but no divisibility

In this section we focus on the more classical problem with preemption but without the divisible load assumption. We show that combining the approach of the previous Section with the work of Lawler and Labetoulle [7] leads to a polynomial-time algorithm to solve this problem. Note that, for this exact problem, Bender, Chakrabarti, and Muthukrishnan stated in [2] the existence of a fully polynomial time approximation scheme (FPTAS). We do not know whether since that publication someone has already shown that this problem can be solved in polynomial time.

Following the work of Gonzalez and Sahni [5], Lawler and Labetoulle present in [7] a scheme to build in polynomial-time a preemptive schedule of makespan C_{obj} for a set of jobs $J_1, ..., J_n$ of null release dates ($\forall j, r_j = 0$), under the condition that Linear System (4) has a solution. This system simply states that all jobs must be fully processed (Equation (4a)), that the whole processing of a job cannot take a time larger than C_{obj} (Equation (4b)), and that the whole utilization time of a machine cannot be longer than a time C_{obj} (Equation (4b)). Obviously, these constraints must be satisfied by any preemptive schedule whose makespan is no longer than C_{obj}. The result obtained by Lawler and Labetoulle shows that such a schedule exists if, only if, this set of constraints has a solution.

$$\begin{cases} \text{(4a)} \quad \forall j, \quad \displaystyle\sum_{i=1}^{m} \alpha_{i,j} = 1 \\ \text{(4b)} \quad \forall j, \quad \displaystyle\sum_{i=1}^{m} \alpha_{i,j} \cdot c_{i,j} \leqslant C_{\mathrm{obj}} \\ \text{(4c)} \quad \forall i, \quad \displaystyle\sum_{j=1}^{n} \alpha_{i,j} \cdot c_{i,j} \leqslant C_{\mathrm{obj}} \end{cases} \tag{4}$$

Our problem is slightly more general in that we allow arbitrary release dates. Additionally, our objective is to minimize the maximum weighted flow rather than the

makespan. Let us consider a maximum weighted flow objective F_{obj}. As we did in Section 4.3.1, we use this objective value to define for each job J_j a deadline $\bar{d}_j(F_{\text{obj}}) = r_j + F_{\text{obj}}/w_j$. As before, the set of release dates and deadlines defines a set of epochal times which, in turn, defines a set of time intervals that we denote by $I_1, \ldots, I_{n_{\text{int}}(F_{\text{obj}})}$.

Then, we claim that there exists a preemptive schedule whose maximum weighted flow is no greater than F_{obj} if, and only if, Linear System (5) has a solution. Linear System (5) simply states that:

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

$$\begin{cases} \text{(5a)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \\ \text{(5b)} & \forall t, \forall j, \quad \sum_i \alpha_{i,j}^{(t)}.c_{i,j} \leqslant \sup I_t - \inf I_t \\ \text{(5c)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leqslant \sup I_t - \inf I_t \\ \text{(5d)} & \forall i, \forall j, \forall t, \quad r_j \geqslant \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(5e)} & \forall i, \forall j, \forall t, \quad \bar{d}_j \leqslant \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \end{cases} \tag{5}$$

Any preemptive schedule whose maximum weighted flow is no greater than F_{obj} must obviously satisfy Linear System (5). Conversely, suppose that Linear System (5) has a solution. Then, following Lawler and Labetoulle [7], we note that for each interval I_t, the system effectively decomposes into a linear sub-system that is exactly equivalent to Linear System (4) where the objective is the length of the time interval ($C_{\text{obj}} = \sup I_t - \inf I_t$). Therefore, starting from a solution of Linear System (5) we use the polynomial-time reconstruction scheme of Lawler and Labetoulle to build a preemptive schedule on each of the time intervals I_t. The concatenation of these partial schedules gives us a solution to our problem.

Thus far, we have shown that we are able to check the feasibility of a specific objective value for maximum weighted flow in polynomial time. Moreover, if such an objective is feasible a schedule that achieves this maximum weighted flow can also be built in polynomial time. To finally solve our problem, we recall the methodology presented in Section 4.3.1:

Linear System (5) can be used to search for a solution in a range of objective values, defined by consecutive *milestones*, over which the linear system is valid (i.e., the relative order or task release dates and deadlines do not change). Similarly, a binary search over the possible milestone ranges enables us to find the optimal solution in polynomial time.

5 Conclusion

We have initially shown experimentally that the divisible load framework is suitable for our practical implementation. In this framework, we then presented a polynomial-time algorithm to solve the theoretical off-line scheduling problem. Solving the off-line problem not only gives us a bound against which we can compare actual on-line solutions, it also suggests on-line scheduling strategies that are likely to prove efficacious. In some preliminary simulations, we see that a simple on-line adaptation of our off-line algorithm, enhanced by a simple preemption scheme, produces better schedules than classical scheduling heuristics like Minimum Completion Time, with respect to our objectives. Based on these promising results, we plan to further investigate the on-line version of our problem. Furthermore, we plan to implement a scheduler in a distributed environment running the GriPPS biological sequence comparison application.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

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

[4] A. E. Darling, L. Carey, and W. chun Feng. The Design, Implementation, and Evaluation of mpiBLAST. In *Proceedings of ClusterWorld 2003*, 2003.

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

[6] J. Labetoulle, E. L. Lawler, J. K. Lenstra, and A. H. G. R. 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.

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

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

[9] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

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

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

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