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

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

Alan Su
LIP, ENS Lyon, France
Alan.Su@ens-lyon.fr

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

Abstract

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

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 [11] sequence comparison algorithms have been realized for various computational environments (e.g., [4, 5, 10]). 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. 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 [3, 7] 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. In the following sections we describe 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 (Section 6) and in the more classical framework with task preemption (Section 7). Before that, Sections 4 and 5 are devoted to preliminary results. Finally, we conclude by discussing our planned extensions to this work in Section 8.

2. Framework

The GriPPS [3, 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 ex-

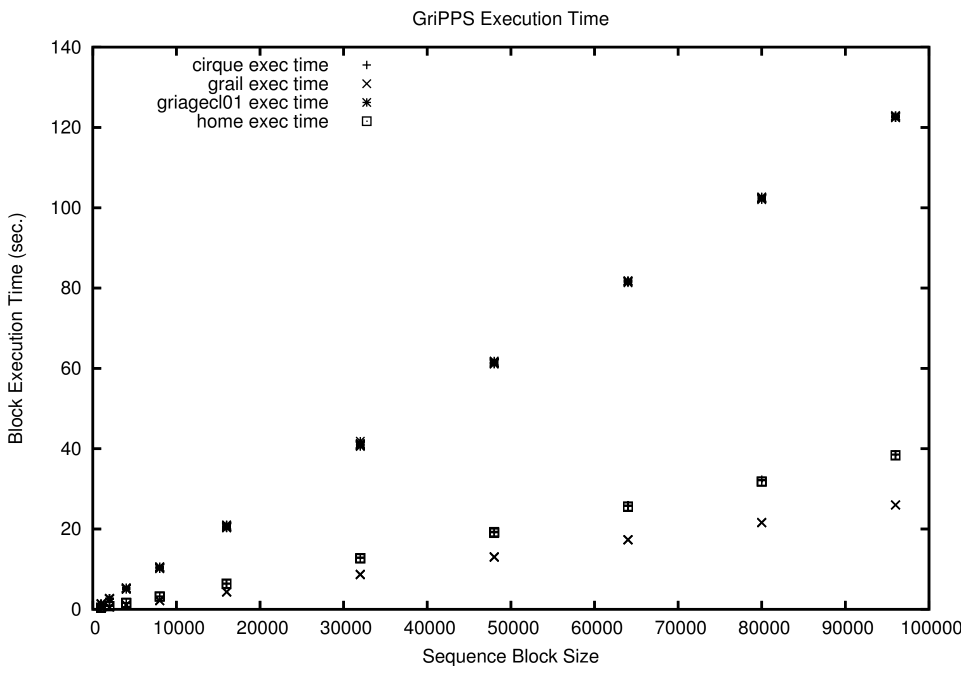


Figure 1. Sequence databank divisibility studies.

periments 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:

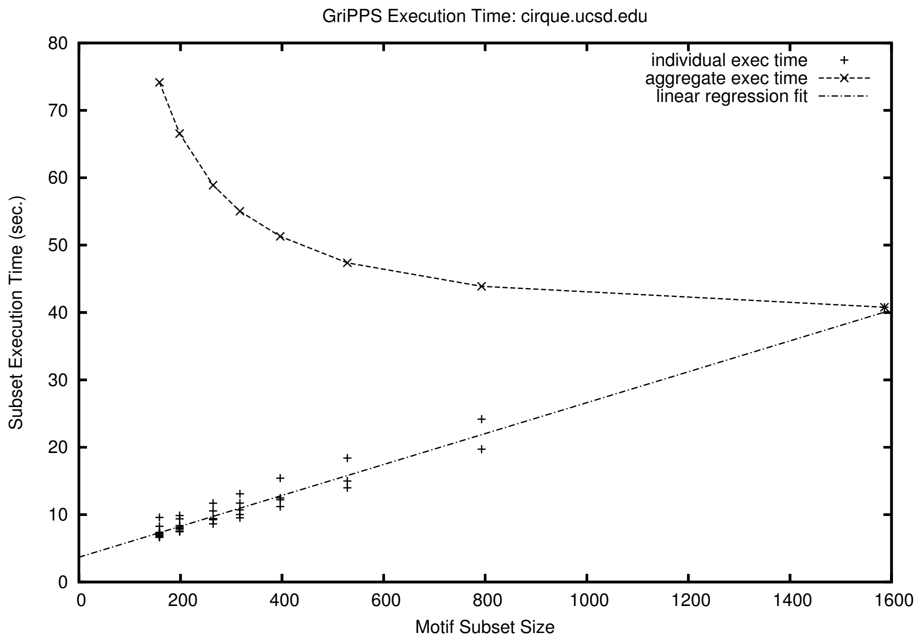
1. **protein databanks**: the reference databases of amino acid sequences, located at fixed locations in a distributed heterogeneous computing platform;
2. **motifs**: compact representations of amino acid patterns that are biologically significant and serve as user input to the application;
3. **sequence 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 two sets of experiments we conducted that demonstrate two properties of the GriPPS applications: workload divisibility and uniform computation.

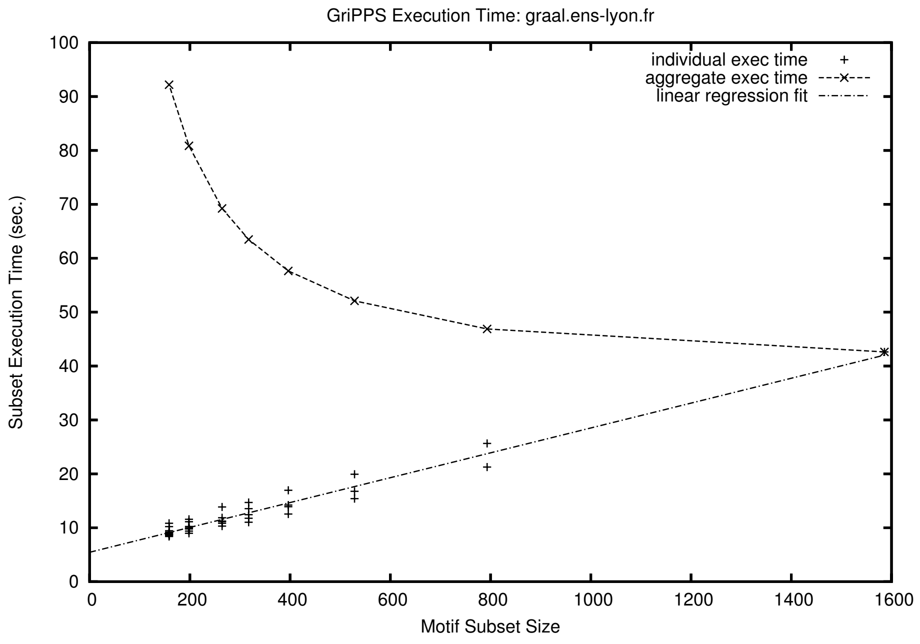
2.1. Divisible Workload

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 1600 motifs and a database of approximately 106,000 protein sequences. We consider a series of subset sizes for the protein database, ranging in size from 1,000 sequences to 96,000 sequences. For each subset size, we construct ten randomly generated sequence database subsets of the appropriate size. 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; each series comprises 100 runs (ten database subsets for each of the ten partition size). We repeated this experiment on a number of different platforms; the data for each machine are presented in Figure 1, which 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 block size and computation time looks nearly perfectly linear and exhibits a low degree of variation. To quantify this correlation, we performed a linear regression analysis on the sequence set partitioning data corresponding to one of our benchmarking machines (cirque.ucsd.edu). This regression indicates an overhead of -0.007 seconds, suggesting that the GriPPS workload exhibits almost no computational overhead and is almost perfectly divisible when considering sequence database partitioning.

In order to evaluate the divisibility of the GriPPS application on the basis of the motif set, we performed a set of experiments based on partitioning the approximately 1600



(a) Motif set divisibility on cirque.ucsd.edu.



(b) Motif set divisibility on graal.ens-lyon.fr.

Figure 2. Motif set divisibility studies.

motifs into subsets of varying size. However, we found that, compared to protein sequences, the complexity of the GriPPS algorithms are more dependent on the structure and contents of the individual motifs. Consequently, we chose subset sizes such that, for a given subset size, we could construct motif subsets that were both disjoint and that, in aggregate, comprised the full motif set. For each block size, we then invoked the GriPPS comparison application to find matches between each motif subset among the entire reference sequence database. The results of two such experiments are presented in Figures 2(a) and 2(b); these results were typical for this experiment repeated on different platforms. In this plot, eight distinct block sizes are chosen; these correspond to partitioning the motif set into 1, 2, 3, 4, 5, 6, 8, and 10 equal-sized but randomly chosen blocks. From the individual data points representing each of these experiments, one can appreciate the fact that execution times vary substantially even between blocks containing the same number of motifs, supporting our finding that motif complexity may vary widely in a given motif set. Note that, in aggregate, the experiments for each block size effectively performed the same work; the results of the application run using a single motif set were identical to those obtained when the motif set was divided into ten subsets. Thus, by taking the sum of computation times for a particular block size, we can fairly assess the effects of partitioning the motif set at that granularity. The aggregate execution time for each of the eight partition sizes clearly indicates a substantial overhead cost associated with partitioning the motif set.

Clearly, there exists 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 appears to exist in the empirical data. To quantify this overhead in the experiments shown in Figure 2, we performed linear regression analyses on the motif set partitioning data. Such analyses serve to project the significance of the computational overhead associated with motif partitioning. The computational overhead was computed to be 3.7 seconds for cirque.ucsd.edu and 5.4 seconds for graal.ens-lyon.fr. For comparison, on cirque.ucsd.edu, an average of 38.5 seconds were needed to compare the entire motif set against a block of 96,000 sequences; clearly such overhead costs are non-negligible.

We also ran a second series of experiments to evaluate the impact of inter-processor communication on application performance. In the GriPPS scenario, the protein sequence databanks are located at fixed nodes, and only the motifs need to be distributed (and the results need to be sent back). We considered data transfer times for the file containing full motif set used in the experiments described above, in a cluster environment based on a gigabit/second interconnection network. In this environment, we use scp to repeatedly transfer the data file (which occupies 5.8 megabytes of disk space) at random intervals. Over 20 iterations, the average data transfer time was 854ms, ranging between 828ms and 880ms. Moreover, the GriPPS developers have indicated that the profiles in this file contain a substantial amount of data that is not used by their sequence comparison algorithm; in one sample motif file we analyzed closely, the important motif entries comprise only 3.5% of the total file size. By transferring only these critical data, and by using other optimizations (e.g., compression), we believe we can further reduce communication overhead costs substantially.

We performed a similar analysis on the transfer times for the GriPPS result output, containing matches between the specified motif set and protein sequences. On the same gigabit/second network infrastructure used in the motif set transfer tests, we considered the result file for a full comparison of 1600 motifs against all 106,000 sequences. The resulting output comprised 36 megabytes of data, requiring between 2.72s and 2.80s to transfer using scp. This provides an indication of a conservative upper bound on the cost of any single transfer operation resulting from this scenario, as any comparison against a subset of the sequence database will necessarily produce fewer matches. However, this magnitude of data may be a consideration as the system is scaled to handle a much greater request volume. It should be noted that the current output format is designed to be human-readable. In a realistic parallel application scenario, such a format would only be generated once the results from all computational tasks are present at the master node. However, even without such optimizations, total execution time is dominated by the computational requirements of this application, for reasonable parameterizations of the GriPPS application; thus, in this study, we neglect data transfer costs.

To summarize, the GriPPS protein databank search application is an example of a *linear divisible workload without communications* due to (i) the linear relationship between the job computation costs and the size of the targeted protein sequence set, and (ii) the negligible communication overheads. We will develop scheduling strategies that take advantage of these properties in Sections 4-7.

2.2. Uniform Computation

A set of tasks is uniform over a set of processors if the relative execution times of tasks over the set of processors does not depend on the nature of the tasks. More formally, for any task T_j, $c_{i,j}/c_{i',j} = k_{i,i'}$, where $c_{i,j}$ is the time needed to process task T_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 task being considered.

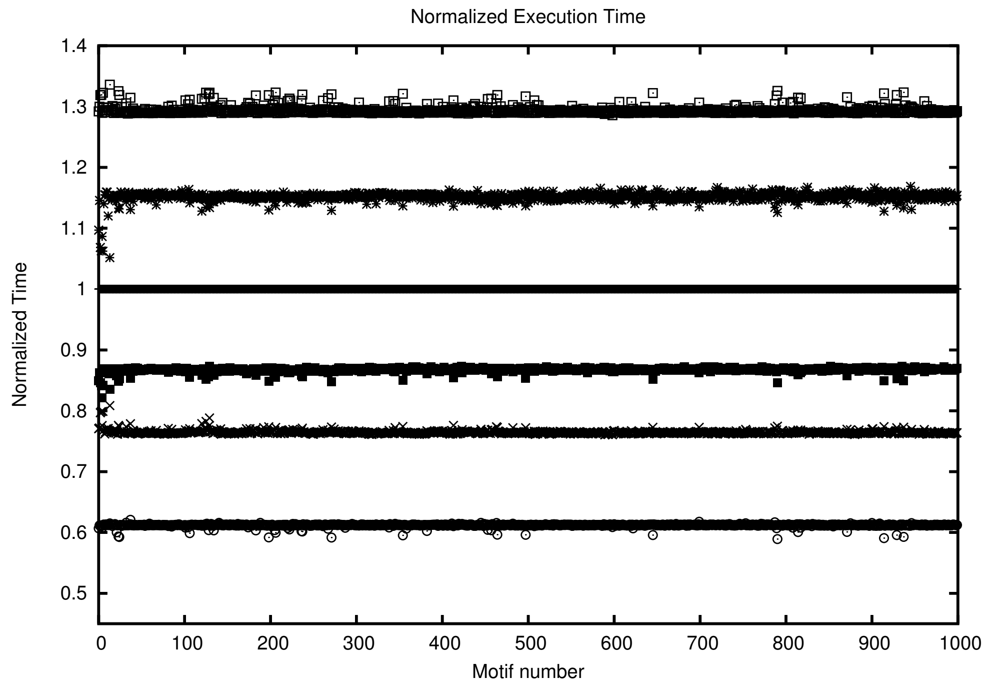


Figure 3. Uniformity study on six different machines.

The results depicted in Figure 1 seem to indicate that the GriPPS comparisons of a motif against a sequence database define a uniform set of tasks: over the range of sequence partition sizes, the execution times on any machine seem to be related to those of any other by a constant multiplicative factor.

To determine if the GriPPS workload is indeed uniform, we conducted a series of experiments to test the performance of the application for a range of motifs on six different machines with varying hardware configurations. No effort was made to ensure that our benchmark processes had exclusive access to the resources, but in general, we observed that the platforms were relatively quiescent. On each machine, we ran a series of comparisons, each searching for instances of a single motif in the full database of 106,000 sequences. This experiment is run for the first 1000 motifs in the motif set. To serve as a baseline performance measure in the evaluation of the uniformity property, we arbitrarily used one of our machine as our reference machine. The entire 1000-motif series was executed from three to ten times on each of our six target machines. The execution times from these experiments were averaged according to motif, after the extremal execution times for each motif were removed. For each motif, the resulting average execution times were then normalized to the average execution time on the reference machine. These results are presented in Figure 3. The normalized execution time of the reference machine is, by definition, 1.0 for each motif, and of the five remaining machines, three exhibit shorter execution times and two exhibit longer execution times. In general, the data 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 the reference machine for that same motif. This trend supports the hypothesis of uniformity. To quantify this assertion, Table 1 presents basic statistical analyses on the normalized performance data of the five non-reference machines shown in Figure 3.

Machine number	Mean Normalized Execution Time	Standard Deviation
1	0.61180	0.00259
2	0.76472	0.00339
3	0.86732	0.00378
4	1.15149	0.00833
5	1.29377	0.00608

Table 1. Statistical analyses of normalized execution data.

Based on our experimental findings, we propose a theoretical framework to study scheduling problems in the context of series of requests on a set of databanks distributed across an heterogeneous computational platform. In particular, the following section presents formal models for the divisible workloads motivated by this class of applications, and the distributed heterogeneous platforms we are targeting.

3. Platform and application model

3.1. Notations

Formally, an instance of our problem is defined by n jobs, J_1, ..., J_n and m machines (or processors), M_1, ..., M_m. The job J_j arrives in the system at time r_j (expressed in seconds), which is its release date; we suppose jobs arrive ordered by increasing release dates. Each job is assigned a *weight* or *priority* w_j.[1] $c_{i,j}$ 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. 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 2.2, 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 scheduling problem formulation (i.e., unrelated machines).

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

1 In this paper, we solve our problem for arbitrary weights and we do not discuss the definition of these weights. For an example of weight definition, see the remarks below about maximum *stretch* minimization.

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 machines.

3.3. 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 time* (also called *response time*), i.e., the time their jobs spend 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 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 inversely proportional to its size $w_j = 1/W_j$. 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 will show that we are able to solve the maximum weighted flow scheduling problem on unrelated machines in polynomial time, either under the divisible load hypothesis or in the more classical framework of preemption (with migration).

3.4. Off-line scheduling

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 dependences, 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 describes the solution of the makespan minimization problem in the divisible load framework for our application model. We then discuss in Section 5 the problem of deadline scheduling and its polynomial-time solution in the same application context. These results are then extended in Section 6, 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 7.

4. 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 machine M_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 machine cannot be used longer than the duration of this time interval (Equation (1b));
3. *end of schedule*: during the last interval, I_n, any machine 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 greater than or equal to the release date of the final job plus Δ_n, the maximum time any machine works after this date, that is, during the final interval, I_n. Hence, the given objective function represents the makespan.

$$\text{MINIMIZE} \quad C_{\max} = r_n + \Delta_n \quad ,$$
$$\text{UNDER THE CONSTRAINTS}$$
$$\begin{cases} \text{(1a)} \quad \forall i, \forall j, \forall t, \quad r_j \geq \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\ \text{(1b)} \quad \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \leq \sup I_t - \inf I_t \\ \text{(1c)} \quad \forall i, \quad \sum_j \alpha_{i,j}^{(n)}.c_{i,j} \leq \Delta_n \\ \text{(1d)} \quad \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1 \end{cases} \tag{1}$$

Any feasible solution to Linear Program (1) gives us a straightforward optimal solution to the makespan minimization problem: during any time interval I_t, and on any machine M_i, we can greedily schedule in any order 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.*

5. 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 occur; there are between 2 (when all jobs are released at the same date and have the same deadline) and $2n$ (when all job release dates and 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 machines 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 cannot be processed after its deadline (Equation (2b));
3. *resource usage*: during a time interval, a machine 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)).

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

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 for any time interval I_t, and on any machine M_i, the job fractions $\alpha_{i,j}^{(t)}$ can be scheduled in any order.

6. Minimizing the maximum weighted flow (in the divisible load framework)

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

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

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 rational values will not terminate. 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. However, as we now show, we can adapt our search to always find the optimal in polynomial time.

6.2. 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.

6.2.1. 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 $[\mathcal{F}_1, \mathcal{F}_2]$ ($\mathcal{F}_1 \le \mathcal{F} \le \mathcal{F}_2$), and with the minimization of $\mathcal{F}$ as the objective. This gives us System (3).

$$\begin{array}{l}
\text{MINIMIZE} \quad \mathcal{F}\ , \\
\text{UNDER THE CONSTRAINTS} \\
\left\{\begin{array}{ll}
\text{(3a)} & \mathcal{F}_1 \le \mathcal{F} \le \mathcal{F}_2 \\
\text{(3b)} & \forall i, \forall j, \forall t, \quad r_j \ge \sup I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\text{(3c)} & \forall i, \forall j, \forall t, \quad \bar{d}_j \le \inf I_t \Rightarrow \alpha_{i,j}^{(t)} = 0 \\
\text{(3d)} & \forall t, \forall i, \quad \sum_j \alpha_{i,j}^{(t)}.c_{i,j} \le \sup I_t - \inf I_t \\
\text{(3e)} & \forall j, \quad \sum_t \sum_i \alpha_{i,j}^{(t)} = 1
\end{array}\right.
\end{array} \tag{3}$$

6.2.2. 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*.[2] In our problem, there are at most n distinct release dates and as many distinct deadlines. Thus, there are at most $\frac{n(n-1)}{2}$ milestones at which a deadline function coincides with a release date. There are also at most $\frac{n(n-1)}{2}$ milestones at which two deadline functions coincides (two affine functions intersect in at most one point). Let n_q be the number of distinct milestones. Then, $1 \le n_\text{q} \le n^2 - n$. We denote by $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_\text{q}}$ the milestones ordered by increasing values. To solve our problem we just need to perform a binary search on the set of milestones $\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_{n_\text{q}}$, each time checking whether System (3) 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 $[\mathcal{F}_{n_\text{q}}, +\infty[$). Hence, we have the following theorem:

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

7. 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 [9] leads to a

2 In [8], Labetoulle, Lawler, Lenstra, and Rinnoy Kan call such a value a "critical trial value".

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 [6], Lawler and Labetoulle present in [9] 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:

1. all jobs must be fully processed (Equation (4a));
2. the whole processing of a job cannot take a time larger than C_{obj} (Equation (4b));
3. 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 constructive result obtained by Lawler and Labetoulle shows that such a schedule exists if, and only if, this set of constraints has a solution.

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

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 $\mathcal{F}_{\text{obj}}$. As we did in Section 6.1, we use this objective value to define for each job J_j a deadline $\bar{d}_j(\mathcal{F}_{\text{obj}}) = r_j + \mathcal{F}_{\text{obj}}/w_j$. 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 $I_1, \ldots, I_{n_{\text{int}}(\mathcal{F}_{\text{obj}})}$.

Then, we claim that there exists a preemptive schedule whose maximum weighted flow is no greater than $\mathcal{F}_{\text{obj}}$ 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 as, in the current framework, a job cannot be simultaneously processed by two different machines (Equation (5b) which corresponds to Equation (4b));
3. the 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)).

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

Any preemptive schedule whose maximum weighted flow is no greater than $\mathcal{F}_{\text{obj}}$ must obviously satisfy Linear System (5). Conversely, suppose that Linear System (5) has a solution. Then, following Lawler and Labetoulle [9], we note that the whole system effectively decomposes into a set of linear sub-systems, one for each of the time intervals, and that the sub-system corresponding to interval I_t is exactly equivalent to Linear System (4) where the objective is the length of the time interval (i.e., $C_{\text{obj}} = \sup I_t - \inf I_t$). 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 for each of the time intervals I_t. The concatenation of these partial schedules gives us a solution to our problem.

Thus far, we have shown that we are able to check the feasibility of a specific objective value for maximum weighted flow in polynomial time. Moreover, if such an objective is feasible a schedule that achieves this maximum weighted flow can also be built in polynomial time. To finally solve our problem, we recall the methodology presented in Section 6.1: Linear System (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 of task release dates and deadlines do not change). Similarly, a binary search over the possible milestone ranges enables us to find and build an optimal solution in polynomial time.

8. 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 of maximum weighted flow minimization. Solving the off-line problem not only gives us a bound against which we will be able to compare actual on-line solutions, it also suggests on-line scheduling strategies that are likely to prove efficacious. We now plan to 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. In such a practical setting we will have to take into account the cost of preemptions, what we have not done so far.

Acknowledgements

The authors would like to acknowledge the cooperation of the GriPPS project team at the Institut de Biologie et Chimie des Protéines (IBCP) in Lyon, France. In particular, we would like to thank Nicolas Lacorne, Christophe Blanchet, and Gilbert Deléage, for allowing us access to the code that was used to validate our application model, and for detailed discussions concerning realistic usage scenarios for the GriPPS system.

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] 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.

[4] 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.

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

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

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

[8] J. Labetoulle, E. 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] 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.

[10] 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.

[11] 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.

Biographies

Arnaud Legrand was born in 1977 in Créteil, France. He obtained a PhD thesis from École normale supérieure de Lyon in 2004. He is currently a full researcher from CNRS in the Computer Science Laboratory ID at Grenoble. His main research interests are parallel algorithm design and scheduling techniques for heterogeneous platforms, performance evaluation and simulation/emulation of large heterogeneous distributed computing platforms.

Alan Su completed his Ph.D. at the University of California, San Diego in 2003, advised by Professors Henri Casanova and Francine Berman. Since January 2004, he is a post-doctoral researcher in the GRAAL research group at the École normale supérieure in Lyon, France. His research focuses on distributed and parallel computing techniques that enable high-performance scientific applications in computational cluster and grid environments.

Frédéric Vivien was born in 1971 in Saint-Brieuc, France. He obtained a PhD thesis from École normale supérieure de Lyon in 1997. From 1998 to 2002, he had been an associate professor at Louis Pasteur University of Strasbourg. He spent the year 2000 working in the Computer Architecture Group of the MIT Laboratory for Computer Science. He is currently a full researcher from INRIA. His main research interests are scheduling techniques, parallel algorithms for clusters and grids, and automatic compilation/parallelization techniques.