Models for scheduling on large scale platforms: which policy for which application?*

Pierre-François Dutot, Lionel Eyraud, Grégory Mounié and Denis Trystram
ID-IMAG
51 avenue Jean Kuntzmann
38330 Montbonnot Saint Martin, France

Abstract

In the recent years, there was a huge development of low cost large scale parallel systems. The design of efficient parallel algorithms has to be reconsidered by the influence of new parameters of such execution supports (namely, clusters of workstations, grid computing and global computing) which are characterized by a larger number of heterogeneous processors, often organized by hierarchical subsystems.

Alternative computational models have been designed in order to take into account new characteristics. Parallel Tasks model – PT in short – (i.e. tasks that require more than one processor for their execution) is a promising alternative for scheduling parallel applications, especially in the case of slow communication media. The basic idea is to consider the application at a rough level of granularity. Another way of looking at the problem (which is somehow a dual view) is the Divisible Load model (DL) where an application is considered as a collection of a large number of elementary – sequential – computing units that will be distributed among the available resources.

As the main difficulty for scheduling in actual systems comes from handling efficiently the communications, these two new views of the problem allow us to consider them implicitly or to mask them, thus leading to more tractable problems.

This paper aims first at presenting some examples of approximation algorithms for parallelizing applications for the PT model with a special emphasis on new execution supports. Then, we will show how to mix these results with the DLT model in order to integrate them into the previous model for managing the resources of an actual computational grid composed by more than 600 machines built in Grenoble (CiGri project).

1. Introduction

1.1. Parallel Processing today

In the Parallel Processing area, scheduling is a crucial problem for determining the starting times of the tasks and the processor locations. Many theoretical studies were conducted [2] and some efficient practical tools have been developed for old generation shared-memory systems [9, 19].

Scheduling in modern parallel and distributed systems is much more difficult because of new characteristics of these systems. These last few years, super-computers have been replaced by collections of large number of standard components, physically far from each other and heterogeneous [7]. The need of efficient algorithms for managing these resources is a crucial issue for a more popular use. Today, the lack of adequate software tools is the main obstacle for using these powerful systems in order to solve large and complex actual applications.

The classical scheduling algorithms that have been developed for parallel machines of the nineties are not well adapted to new execution supports. The most important factor is the influence of communications. The first attempt that took into account the communications into computational models was to adapt and refine existing models into more realistic ones (delay model with unitary delays [12], LogP model [6]). However, even the most elementary problems are already intractable [18], especially for large communication delays. The other characteristics of the new execution supports are heterogeneity of processors or communication media, several levels of hierarchy (from SMP nodes to clusters and grids), versatility of the system components (some nodes can appear or disappear, new jobs can be created at any moment depending of the results of a job, etc.).

Our view of the problem is to consider the notion of *light grid* as a collection of few clusters in a same geographical area. It is an intermediate step for a better understanding of

general grid and global computing. we propose a pragmatic approach which is based on several years of experience using a 225 PC cluster at IMAG and the regional grid CiGri gathered more than 600 machines [5]. We describe briefly in the next section the corresponding architectural model.

1.2. Description of the Platform model

The target execution support that we consider here is a few clusters composed each by a collection of a medium number of SMP or simple PC machines (typically several tenth or several hundreds of nodes). Such a system may be highly heterogeneous between clusters (different kind of processors, different numbers of processors, different Operating Systems, etc.), but weakly heterogeneous inside each cluster (different generations of processors running under the same Operating System with different clock speeds). No specific topology is assumed, but the interconnection network is fast and may be hierarchical.

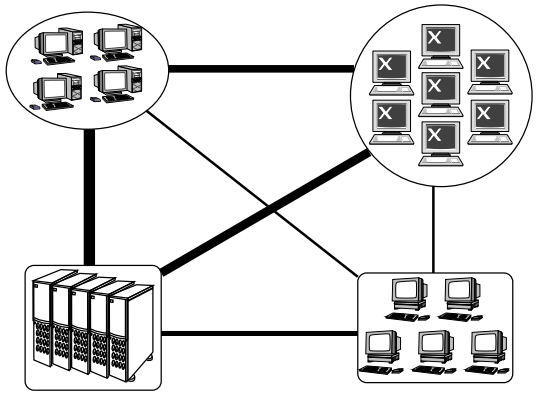

Figure 1. A light grid.

The submissions of jobs is done by some specific nodes by the way of several priority files. No other submission is allowed. Each cluster is administrated by a separate system engineer but of course, these internal managing rules have been established after many exchanges between the communities.

1.3. How to manage the resources?

There is no global consensus today for an universal way of looking at the resource management problem on grids. Adequate computational models have to be developed for designing and analyzing scheduling algorithms.

The delay models, based on explicit handling of communications, should be forgotten because of their intrinsic intractability. Two alternative models have been proposed, namely the Divisible Load Tasks model (DLT) and the Parallel Tasks model (PT).

We believe that there is no issue for determining a standard solution for managing the resources. As the objectives may be different from one community to another one, it seems impossible to formalize the global problem as a classical combinatorial optimization problem. Thus, leading to well-founded algorithms with some guaranties.

2. Alternative models

2.1. Divisible Load – DLT

A Divisible Load Task can be seen as a (usually large) set of computations that can be partitioned in every possible way, each part being completely independent of the other parts. This model was first introduced in [4] for the processing of big data files.

As each part has to be completely independent, the jobs modeled with the DLT model cannot have data dependencies or communication within the task. With the partitioning property, the atomic computations of the job have to be very small with respect to the total work (fine grain). Since the introduction of this model, many kinds of applications have been considered as Divisible Load Tasks, such as parametric executions or image and signal processing.

The DLT model is well suited for heterogeneous platforms, and slow communications, as no communications occur after the distribution of the task. The difficulty of scheduling lies in the distribution of the task to the available processors. This distribution can be made in one, several rounds or dynamically with a work stealing strategy [3]. Simple problems as the single round distribution on processors connected by a common bus are polynomial, but the complexity becomes quickly NP-hard with more general network topologies.

At the end of the computation, if we are for example searching something in a database there is only one processor which have to send back data. However, if all the data processing produces output, the communications gathering the results can be done as a mirror image of the data distribution.

2.2. Parallel Tasks – PT

Informally, a Parallel Task (PT) is a *task* that gathers elementary operations, typically a numerical routine or a nested loop, which contains itself enough parallelism to be executed by more than one processor.

We consider PT as independent jobs (applications) submitted in a multi-user context. Usually, new PT are submitted at any time (on-line). The time for each PT can be estimated or not (clairvoyant or not) depending on the type of applications. We will consider mainly the first case in this paper: we have an estimation of the characteristics of

the submitted jobs (expected running times, parallel profile – at least qualitatively, etc.).

The PT model seems particularly well-adapted to grid and global computing because of the intrinsic characteristics of these new types of supports: large communication delays which are considered implicitly and not explicitly like they are in all standard models, the hierarchical character of the execution support which can be naturally expressed in PT model. The heterogeneity of computational units or communication links can also be considered by uniform or unrelated processors for instance.

We usually distinguish between three types of Parallel Tasks (PT):

- *Rigid* jobs when the number of processors to execute the PT is fixed a priori. In this case, the PT can be represented as a rectangle in a Gantt chart. The allocation problem corresponds to a strip-packing problem [13].

- *Moldable* jobs when the number of processors to execute the PT is not fixed but determined before the execution. As in the previous case this number does not change until the completion of the PT.

- *Malleable* jobs when the number of processors may change during the execution (by preemption of the tasks or simply by data redistributions).

For historical reasons, most of submitted jobs are rigid. However, intrinsically, most parallel applications are moldable. An application developer does not know in advance the exact number of processors which will be used at run time. Moreover, this number may vary with the input problem size or number of nodes availability. This is also true for many numerical parallel library. Most of the main restrictions are the minimum number of processors that are needed because of time, memory or storage constraints.

The main restriction in a systematic use of the moldable character is the need for a practical and reliable way to estimate (at least roughly) the parallel execution time as function of the number of processors. Most of the time, the user has this knowledge but this is an inertia factor against the more systematic use of such models. Most parallel programming tools or languages have some malleability support, with dynamic addition of processing nodes support. Modern advanced parallel programming environments, like Condor, Globus or Mosix implement advanced capabilities, like resilience, preemption, migration, or at least the model allows us to implement these features.

Malleability is much more easily usable from the scheduling point of view but requires advanced capabilities from the runtime environment, and thus restrict the use of such environments and their associated programming models. In the near future, moldability and malleability should be used more and more. We will not consider malleability here.

3. Optimization criteria

The main objective function used historically is the *makespan*. This function measures the ending time of the schedule, i.e., the latest completion time over all the tasks. However, this criterion is valid only if we consider the tasks altogether and from the viewpoint of a single user. If the tasks have been submitted by several users, other criteria can be considered. Let us review briefly various possible criteria usually used in the literature:

- Minimization of the *makespan* ($C_{max} = max(C_j)$ where the completion time C_j is equal to $\sigma(j) + p_j(nbproc(j))$). p_j represents the execution time of task j, σ function is the starting time and $nbproc$ function is the processor number (it can be a vector in the case of specific allocations for heterogeneous processors).

- Minimization of the average completion time (ΣC_i) [16, 1] and its variant weighted completion time ($\Sigma \omega_i C_i$). Such a weight may allow us to distinguish some tasks from each other (priority for the smallest ones, etc.).

- Minimization of the mean *stretch* (defined as the sum of the difference between completion times and release dates: $\Sigma C_i - r_i$). In an on-line context it represents the average response time between the submission and the completion.

- Minimization of the maximum stretch (i.e. the longest waiting time for a user).

- Maximum throughput (or steady state) defined as the maximum number of elementary tasks to execute in a given amount of time or for asymptotically long times. It is well-suited for some types of jobs like parametric computations.

- Minimization of the tardiness. Each task is associated with an expected due date and the schedule must minimize either the number of late tasks, the sum of the tardiness or the maximum tardiness.

- Other criteria may include rejection of tasks or normalized versions (with respect to the workload) of the previous ones.

4. Some results about Parallel Tasks

We concentrate in this section on the PT model. We will show some interesting results that can be combined together in order to construct realistic scheduling algorithms.

In the PT model, communications are considered by a global *penalty* factor which reflects the overhead for data distributions, synchronization, preemption or any extra factors coming from the management of the parallel execution. The penalty factor implicitly takes into account some constraints, when they are unknown or too difficult to estimate formally. It can be determined by empirical or theoretical studies (benchmarking, profiling, performance evaluation through modeling or measuring, etc.).

We consider first the case of a single cluster. Independent jobs have been submitted to a file and are ready to be executed. More formally, we consider the off-line scheduling of a set of n independent moldable jobs on m identical processors for minimizing the makespan. Most of the existing methods for solving this problem have a common geometrical approach by transforming the problem into 2 dimensional packing problems. It is natural to decompose the problem in two successive phases: determining first the number of processors for executing the jobs, then solve the corresponding scheduling problem with rigid jobs.

4.1. A good off-line approximation algorithm

We recall briefly the principle on the best known algorithm for solving this problem [8]. The idea is to determine the job allocation with great care in order to fit them into a particular packing scheme that is inspired from the shape of the optimal one.

The MRT algorithm has a performance ratio of $3/2 + \epsilon$ [8]. It is obtained by stacking two shelves of respective sizes λ and $\frac{\lambda}{2}$ where λ is a guess of the optimal value C^*_{max}. This guess is computed by a dual approximation scheme [11]. A binary search on λ allows us to refine the guess with an arbitrary accuracy ϵ.

The guess λ is used to bound some parameters on the tasks. We give below some constraints that are useful for proving the performance ratio. In the optimal solution, assuming $C^*_{max} = \lambda$:

- $\forall j, p_j(nbproc(j)) \leq \lambda$.
- $\sum w_j(nbproc(j)) \leq \lambda m$.
- When two tasks share the same processor, the execution of one of these tasks is lower than $\frac{\lambda}{2}$. As there are no more than m processors, less than m processors are used by the tasks with an execution time larger than $\frac{\lambda}{2}$.

This algorithm is the basis of an on-line algorithm described in the next section.

4.2. On-line batch scheduling

An important characteristic of the new parallel and distributed systems is the versatility of the resources: at any moment, some processors (or groups of processors) can be added or removed. On another side, the increasing availability of the clusters or collections of clusters involved new kind of data intensive applications (like data mining) whose characteristics are that the computations depend on the data sets. The scheduling algorithm has to be able to react step by step to arrival of new tasks and thus, off-line strategies can not be used. Depending on the applications, we distinguish two types of on-line algorithms, namely, clairvoyant on-line algorithms when most parameters of the Parallel Tasks are known as soon as they arrive, and non-clairvoyant ones when only a partial knowledge of these parameters is available.

Most of the studies about on-line scheduling concern independent tasks, and more precisely the management of parallel resources. We invite the ready to look at the survey [15] for more details about on-line algorithms. In this section, we consider the clairvoyant case, where an estimate of the task execution time is known.

We recall first a generic result about *batch scheduling*. In this context, the jobs are gathered into sets (called batches) that are scheduled together. All further arriving tasks are delayed to be considered in the next batch. This is a nice way for dealing with on-line algorithms by a succession of off-line problems. We will use the result of Shmoys et al. [17] which proposed how to adapt an algorithm for scheduling independent tasks without release dates (all tasks are available at date 0) with a performance ratio of ρ into a batch scheduling algorithm with unknown release dates with a performance ratio of 2ρ.

Now, using the off-line algorithm with a performance ratio of $3/2 + \epsilon$, it is possible to schedule moldable independent tasks with release dates with a performance ratio of $3 + \epsilon$ for C_{max}. The algorithm is a batch scheduling algorithm, using the independent tasks algorithm at every phase.

The makespan criterion has not always a clear meaning, especially for very long execution windows. The users usually prefer to have a guaranty that in average, their jobs are performed in the minimum time.

4.3. Batch scheduling for average completion time

Scheduling to minimize the average completion times is very different than scheduling to minimize the makespan. Good scheduling algorithm for one criterion usually have a very big performance ratio for the other criterion. The single machine problem has a polynomial optimal solution which consists of sorting the tasks with increasing sizes and schedule them in this order. In the weighted case, where each task is given a weight (defining its priority), the scheduling is made according to the ratio time/weight.

In the off-line multi-processor case, scheduling with batches (or shelves) allows us to return to this simple single machine problem. Each batch has a time length and a weight (the sum of the weight of their tasks) and finding the optimal order of batches is exactly the single machine problem. Schwiegelshohn et al. [14] proposed for rigid PTs to use shelves (where all the tasks start at the same time) filled with tasks of approximately the same length (shelves sizes are powers of 2). The performance ratio is 8 for the unweighted case and 8.53 for the weighted case.

The shelves here were just filled with a first fit algorithm. We will see that this ratio can be improved using more complex scheduling algorithms within batches instead of stacking tasks on shelves.

4.4. Bi-criteria analysis

As said before, several criteria could be used to describe the quality of a schedule. The choice of which criterion to choose depends on the users view of the problem or the system administrators point of view.

However, one could wish to take advantage of several criteria in a single schedule. We present here such an analysis for the two most popular criteria (which are somehow antagonistic). With the makespan and the sum of weighted completion times, it is easy to find examples where there is no schedule reaching the optimal value for both criteria. We can try to study how far the solution of a schedule is from the optimal one for each criterion. In this section, we will present a specific family of scheduling algorithms for independent on-line moldable jobs.

There exists an approach for obtaining a bi-criteria algorithm starting from two algorithms for each criterion. It is also possible to design an ad hoc bi-criterion algorithm just by adapting an algorithm $\mathcal{A}_{C_{max}}$ designed for the makespan criterion [10]. This solution is better and is detailed below.

The main idea is to use algorithm $\mathcal{A}_{C_{max}}$ (with performance ratio $\rho_{C_{max}}$ on the makespan) as a procedure to build a schedule which has a performance guaranty on the sum of the completion times. The makespan algorithm $\mathcal{A}_{C_{max}}$ takes as input a set of (possibly weighted) tasks and a deadline d, and outputs a schedule of length at most $\rho_{C_{max}} d$ with as many tasks as possible (or the maximum weight).

Running this $\mathcal{A}_{C_{max}}$ algorithm iteratively in batches of doubling sizes (d, $2d$, $4d$, $\ldots$) gives a schedule where the total makespan is at most $4\rho_{C_{max}} C^*_{max}$ as the last batch is smaller than $2\rho_{C_{max}} C^*_{max}$. The performance ratio on the sum of completion times is also $4\rho_{C_{max}}$. The technical proofs are in the original article [10].

5. Integration into an actual environment

In this section, we discuss several directions for integrating the previous ideas in order to build an operational light grid management system. A simulated implementation of a variation of the bi-criteria algorithm has been realized, and yields the encouraging results of fig. 2, where the simulation assumed a cluster of 100 machines, parallel and non-parallel jobs, and two criteria C_{max} and $\sum \omega_i C_i$.

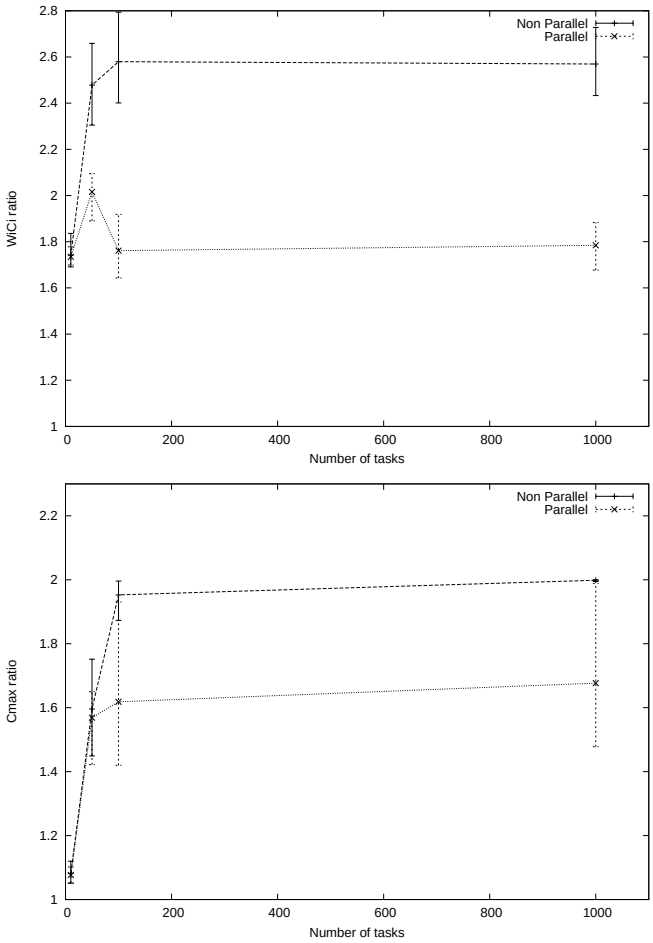

Figure 2. Simulation results.

5.1 Single cluster issues

Rigid Jobs The scheduling algorithms presented in section 4 are targeted for *moldable* jobs. Even though most jobs are intrinsically moldable, some of them need to stay rigid, or at least can not accept every allocation. The reason can be a lack of time to re-code the program to make it moldable, memory constraints which set a minimum number of processors, or the job can be a benchmarking job that requires a preset number of processors. So that means we actually have to deal with a mix of moldable and rigid jobs.

There are different possible ideas to solve this problem. The first trivial idea is to separate rigid and moldable jobs and schedule one category after the other. Another solution is to calculate a-priori an allocation for the moldable jobs, and then apply a rigid scheduling algorithm on the resulting rigid jobs. The last solution is to modify the bi-criteria algorithm in order to schedule each rigid job in the first batch in which it fits. These ideas probably lead to an increased performance ratio.

Reservations An important point for a management system is the ability to perform reservations. This would allow a user to ask for a given number of processors in a given time window. Such a possibility is necessary for demonstration purposes, or in order to set up a wide-area experiment with other computing centers. The scheduling algorithm must then cope with this additional constraint, which makes a certain number of nodes unavailable during a period of time.

Including support for such reservations into a scheduling algorithm is a difficult a problem. A batch algorithm could try to ensure that batch boundaries match the beginning and the end of the reservations, but that would likely be inefficient.

5.2 Dealing with several clusters

This section focuses on the additional problems raised when trying to have several clusters operate together. We will first present the light grid context we are interested on.

The CIMENT project The system we are working on is a part of the CIMENT [5] project, in which the academic computing resources of Grenoble are connected. This results in the realization of a light grid, containing quite heterogeneous machines, more than 500 in total (see fig. 3). The goal of the project is to make different research communities (Numerical Physicists, Astrophysicists, Medical Researchers, Computer Scientists, ...) share their computing resources, leading to an overall better use of these resources.

Joining different communities raises several issues. First, every community has its own behavior, either for historical reasons or for reasons linked to the type of research the community performs. For example, the numerical physicists have long (up to several weeks), sequential jobs to perform, while the computer scientists' jobs are shorter, focusing mainly on debug. Scheduling jobs with such a disparity is an issue in itself.

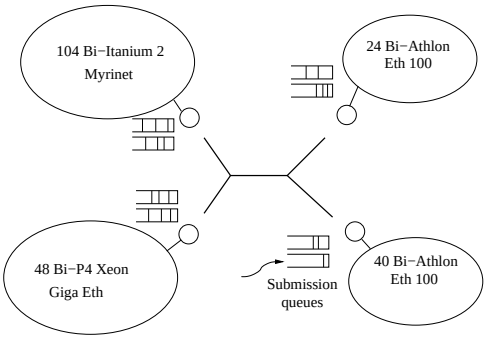

Figure 3. 4 largest clusters of the CIMENT project.

Furthermore, these disparities have implied differences in scheduling choices. It is important to point out that each community has habits about scheduling policy, management system, submission rules, and so on. The light grid management system should try not to disturb these habits by a too large extent.

Another important point is to guarantee a kind of fairness between the different communities. Each computing resource was bought by its respective community because they wanted to use that computing power, so we should make sure that making it available to others does not make them loose too much.

A majority of the jobs submitted in this context are *multi-parametric* jobs. Such a job consists of a large number (up to several hundreds of thousands) of runs of the same program, each having with different parameters. Each run takes a relatively short time to complete, this time being often the same for every run. This kind of jobs are related to the divisible tasks model (see section 2.1). For this kind of jobs, the theory of asymptotic behavior shows that optimal solutions can be computed in polynomial time. This allows the use of these jobs in order to fill the holes in the Gantt chart (using the same idea as conservative backfilling).

We propose two different ways of linking several clusters together. The first one is the current system in use in Grenoble, and the second version is rather an attempt to address the problem more globally.

Centralized In the first vision of this problem, each cluster keeps its own submission system used only for jobs that are to be processed locally. Additionally, there is a centralized server to which all grid jobs are submitted. In this setting, grid jobs are only multi-parametric jobs, which the centralized server submits on the local clusters in order to fill the holes of their respective schedules. This is achieved through the notion of *best-effort* jobs: the local scheduler gives no warranty that the job will be finished. If a locally submitted job requires a processor currently in use by a best-effort job, the latter will be killed. The central server then has to submit it once again. Since there are a large number of relatively small runs, the cost of killing one of them is not too big. Furthermore, this ensures that local users of the clusters will not be disturbed by grid jobs: they have the same submission interface as before and cannot have their job delayed by a grid job.

Decentralized In this vision, all jobs – grid and local ones – are submitted to local scheduling systems. These systems then have the possibility to exchange work in order to balance the load. The protocol for exchanging work still has to be defined, but it would have to take care of both fairness and performance issues at the same time. There are several directions to address this problem: graph coupling which would aim at minimizing data transfers, an economical approach which would have each cluster try to optimize its own jobs, consensus-driven algorithms, view it as a big global optimization problem, ...

6. Conclusion

We presented in this paper our view of how to manage a grid. We developed most the pieces of the puzzle (classical approach), we are now focusing on the hard problem of integrating all parts together. Most interesting parts to integrate are: fixed reservations, rigid tasks, and support for light grids.

We do not believe that it is possible to formalize it as a global combinatorial optimization problem, we focus on alternative methods at the interface of existing local methods.

References

[1] F. Afrati, E. Bampis, A. V. Fishkin, K. Jansen, and C. Kenyon. Scheduling to minimize the average completion time of dedicated tasks. *Lecture Notes in Computer Science*, vol. 1974, 2000.

[2] J. Błażewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling in Computer and Manufacturing Systems*. Springer-Verlag, 1996.

[3] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. In S. Goldwasser, editor, *Proceedings: 35th Annual Symposium on Foundations of Computer Science, November 20–22, 1994, Santa Fe, New Mexico*, pages 356–368, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1994. IEEE Computer Society Press.

[4] Y. Cheng and T. Robertazzi. Distributed computation for a tree network with communication delays. *IEEE Trans. on Aerospace and Electronic Systems*, 24(6):700–712, 1988.

[5] Ciment project. http://ciment.ujf-grenoble.fr.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. Schauser, R. Subramonian, and T. von Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, 1996.

[7] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, inc., San Francisco, CA, 1999.

[8] P.-F. Dutot, G. Mounié, and D. Trystram. *Handbook of combinatorics*, chapter 26. CRC Press, to appear.

[9] A. Gerasoulis and T. Yang. PYRROS: static scheduling and code generation for message passing multiprocessors. In *Proceedings of the 6th ACM International Conference on Supercomputing*, pages 428–437. ACM, July 1992.

[10] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22:513–544, 1997.

[11] D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems: theoretical and practical results. *Journal of the ACM*, 34:144–162, 1987.

[12] J. Hwang, Y. Chow, F. Anger, and C. Lee. Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal on Computing*, 18(2):244–257, Apr. 1989.

[13] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[14] U. Schwiegelshohn, W. Ludwig, J. Wolf, J. Turek, and P. Yu. Smart SMART bounds for weighted response time scheduling. *SIAM Journal on Computing*, 28, 1998.

[15] J. Sgall. Chapter 9: On-line scheduling. *Lecture Notes in Computer Science*, 1442:196–231, 1998.

[16] H. Shachnai and J. Turek. Multiresource malleable task scheduling to minimize response time. *Information Processing Letters*, 70:47–52, 1999.

[17] D. Shmoys, J. Wein, and D. Williamson. Scheduling parallel machine on-line. *SIAM Journal on Computing*, 24(6):1313–1331, 1995.

[18] D. Trystram and W. Zimmermann. On multi-broadcast and scheduling receive-graphs under logp with long messages. In S. Jaehnichen and X. Zhou, editors, *The Fourth International Workshop on Advanced Parallel Processing Technologies - APPT 01*, pages 37–48, Ilmenau, Germany, Sept. 2001.

[19] M.-Y. Wu and D. Gajski. Hypertool: A programming aid for message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343, 1990.