A Runtime Scheduling Method for Dynamic and Heterogeneous Platforms

Salah-Salim Boutammine, Daniel Millot and Christian Parrot
GET / INT,
Département Informatique, 91011 Évry, France
Email: { Salah-Salim.Boutammine, Daniel.Millot, Christian.Parrot } @int-evry.fr

Abstract

In this paper, we present a runtime method for scheduling parallel applications on dynamic and heterogeneous platforms. It can be used to schedule parallel applications whose total workload is unknown a priori. It can also handle the heterogeneous and dynamic conditions of execution that are typical of grids. The method delivers the workload through multiple rounds in order to improve communication/computation overlap on an existing on-line algorithm.

1 Introduction

Scientific computing is no longer limited to supercomputers, multiprocessors or clusters of computers. Grids [13] are on the way, and potentially provide very interesting platforms to execute parallel applications. Although scheduling parallel applications on the nodes of a parallel machine has been widely studied, the use of grids requires an adaptation of scheduling techniques. Indeed, the characteristics of the resources in a grid (available computing power or network bandwidth) may change unpredictably, as these resources are not dedicated to one application.

Therefore scheduling methods have to be reconsidered before grids can be commonly used to run parallel applications efficiently [15]. Moreover, scheduling algorithms generally assume all necessary information about the application is available, which is not always the case: for instance, the total workload or the execution cost of the different tasks may be unknown a priori.

A runtime method for scheduling parallel applications on dynamic and heterogeneous platforms is presented in this paper. It that can be used in the dynamic context of grids or when some of the information traditionally used by scheduling algorithms is lacking. It is based on an on-line algorithm from Drozdowski [12]. Actually, we do not consider grid scheduling in all its acceptions as it encompasses resource discovery, reservation management and many other aspects [22]. We assume that a set of resources has been identified and tackle the problem of distributing optimally the tasks of a parallel application on this set of resources, so that the application terminates as soon as possible.

We consider applications that process a finite –but a priori unknown– amount of data independently. The workload of the application is supposed arbitrarily divisible in chunks, where each chunk consists of some amount of data. The same computation is performed on each data chunk, producing its own result without any communication. This scheme corresponds to a lot of applications exhibiting data parallelism, from many different domains [1, 11].

Typically, such an application could result from the parallelization of the slowest stage of a software pipeline in order to get rid of the corresponding bottleneck. Such applications are suitable for the supervisor-worker programming model, with the supervisor distributing workload chunks to the workers, then collecting the corresponding results from them. Clearly, such a parallelization should only be considered when the processing cost for a chunk by a worker dominates the corresponding communication costs between supervisor and worker in a certain sense. We will be clarified this point later on, when appropriate notations have been introduced (see inequality (2) in section 3).

It has to be noted that although we consider so called divisible load, the DLT (Divisible Load Theory [8, 21, 20, 24]) cannot be straightforwardly applied, due to the fact that we suppose that the total workload of the application is not known a priori. For this reason, we have to use on-line algorithms to tackle our scheduling problem.

As for the execution platform, we consider a single-level tree network, with the root node executing the supervisor process and the leaf nodes executing the worker processes. We adopt a one-port communication model [2] without contention, which means that for a fixed node neither two emissions nor two receptions can overlap each other, whereas

one emission can overlap one reception, and computation can overlap communication. We assume that communication costs are affine in the size of chunks.

This paper is organized as follows. Section 2 defines precisely the scheduling problem we consider. Section 3 describes Drozdowski's scheduling algorithm and introduces some notations. Section 4 presents our runtime method. It first gives an overview of the approach then successively states the conditions for the method to succeed, details its various computations and finally compares it with Drozdowski's method. Section 5 presents related work. Section 6 concludes the paper and outlines future work.

2 The scheduling problem

We consider a supervisor-worker model for which the data to be processed are continuously received by the supervisor in an input buffer until the final item is obtained. It is only when this last item is acquired by the supervisor that the total workload of the application happens to be known. We want to minimize the makespan of the application on a set of heterogeneous (and dynamic) resources. As this problem is known to be NP-complete [14] due to affine cost, it can only be dealt with by means of heuristics.

Execution parameters of the target platform, such as available computing power or network bandwidth, vary both in space (heterogeneity) and in time (dynamicity). We assume that we know all past values of these parameters and are unaware of the future ones.

Considering the choice of the one-port communication model, the workers cannot start their work simultaneously: the supervisor has to finish the emission of some chunk to one worker before being able to begin to send a chunk to another one. Thereafter we assume that we know the optimal order the supervisor should apply for the first round when sending their respective first chunk to the workers in order to minimize the makespan [4]. To terminate the execution of the application as soon as possible under this assumption, the computation should start as soon as possible on all the worker nodes, which should then be sent small initial amounts of work in order to quickly start their computation.

When each worker has received a chunk, the execution enters the so-called steady-state phase (as considered in [16]). The main characteristic of this phase is that the total workload is still unknown. It is therefore difficult to work out a schedule for an early termination of the application. On the other hand, it is conceivable to distribute the load in order to keep the computing nodes as busy as possible. If the choice of the computing resources is optimal (i.e. optimal nodes are chosen in optimal proportion), then keeping the selected nodes active minimizes the makespan. The steady-state phase ends when the supervisor gets the final data item to be processed, and the clean-up phase begins.

From this time instant, the problem of scheduling the remaining load is suitable for DLT, as the total workload is now known: namely the amount of data still present in the supervisor input buffer. So, according to the optimality principle, we can try and minimize the makespan by synchronizing the termination of the computation of all the workers. In order to be able to succeed, the supervisor should not have overloaded any worker too much during the steady-state phase, which would prevent a synchonous termination of all workers. Of course, the larger the supervisor input buffer, the earlier the supervisor knows the amount of load that finally remains to be distributed, thus the more likely a synchronous termination of the execution.

Let us sum up the previous discussion. In order to design an optimal schedule for the whole application execution, we need to meet the following objectives:

- during the start-up phase
  distribute small amounts of data as quickly as possible to the workers
- during the steady-state phase
  make full use of the computing resources and keep the load imbalance between the workers under control
- during the clean-up phase
  have all workers terminate at the same time.

We focus on scheduling during the steady-state phase.

3 Drozdowski's scheduling algorithm

In order to address our problem, we considered the On-Line method presented in [12], denoted "OL method" thereafter. Here we give an idea of this adaptive method, initially worked out to reduce the contention at the supervisor due to the one-port communication model. Such contention happens when two worker nodes simultaneously need to send their results to the supervisor. As long as there are still some tasks left, the OL scheduler allocates new chunks to idle workers (the supervisor knows that a worker is idle when it has received the results corresponding to the chunk it had previously sent to the worker). Therefore, the OL method proceeds incrementally, computing the size of the chunk to be sent to a worker for each new round, in order to try and maintain a constant duration τ for the different rounds and thus avoid contention at the supervisor.
The following notations are used throughout the paper:

- N number of workers,
- $\alpha_{i,k}$ size of chunk sent to worker N_i for the k^{th} round,
- $w_{i,k}$ computation cost for a chunk of size 1 of the k^{th} round by worker N_i,

- S_i start-up time for a communication from the supervisor to N_i,
- $c_{i,k}$ transfer cost for a chunk of size 1 of the k^{th} round to worker N_i,
- S'_i start-up time for a communication from N_i to the supervisor,
- $c'_{i,k}$ transfer cost for the result corresponding to a chunk of size 1 of the k^{th} round from N_i to the supervisor.

So we consider communication start-up times S_i and S'_i to be independent from time and from message size.

In the following, parameters $w_{i,k}$, S_i, $c_{i,k}$, S'_i and $c'_{i,k}$ will be called "execution parameters". As mentionned in section 2, the values of $w_{i,k}$, $c_{i,k}$ and $c'_{i,k}$ are only known for each worker N_i and each round k for which the supervisor has received the corresponding results.

The OL method determines $\alpha_{i,k}$ so as to make the distribution asymptotically periodic with period τ, an arbitrarily fixed value, for all the workers. For worker N_i, let $\sigma_{i,j-1}$ be the elapsed time between the begining of the emission of the chunk of its $(j-1)^{th}$ round and the end of the reception of the result corresponding to this chunk. The OL method determines the value of $\alpha_{i,j}$ as follows:

$$\alpha_{i,j} = \alpha_{i,j-1} \cdot \frac{\tau}{\sigma_{i,j-1}}. \quad (1)$$

That is it allocates bigger (resp. smaller) chunks to the workers with higher (resp. lower) performance. Hence, this method can take the heterogeneous nature of computing and communication resources into account, without explicit knowledge of execution parameters such as available computing power or available bandwidth (as equality (1) shows); as Drozdowski states, "the application itself is a good benchmark" [12] (actually the best one).

Lemma 6.1 in [12] shows that, in a static context, with affine cost models for communication, assignment (1) ensures the convergence of $\sigma_{i,j}$ to τ when j increases indefinitely. Moreover, the OL method guarantees that the delay incurred by a worker compared to the others is upper-bounded by τ.

Being an estimation of the asymptotic period used for task distribution, τ is also an upper-bound on the delay between workers. Being able to control this bound makes it possible to minimize the makespan during the clean-up phase.

Due to its incremental nature, the OL scheduling can take the dynamic nature of computing and communication resources into account, as it optimizes the size of chunks for each round according to parameters estimated during the execution. The time scale used for scheduling should then be adapted to the dynamicity of the execution parameters. For instance the distribution of all the data currently in the supervisor input buffer, if there is a lot, could lend to a schedule which might go too far in the future to be accurate, since it could not take possible evolutions of the execution parameters into account. So, we define $\mathring{\tau}$ as the time period during which the execution parameters do not change "significantly", which we assume to know. It suffices that the frequency $1/\tau$ with which the scheduler checks the value of the execution parameters be greater than $1/\mathring{\tau}$; or

$$\tau \leq \mathring{\tau}.$$

As we assume that communication (resp. computation) costs are affine (resp. linear) in the size of chunks we define precisely, for a chunk of strictly positive size α (i.e. $\alpha \in \mathbb{R}^{+*}$) of the j^{th} round, the cost of:

- sending the chunk to $\mathrm{N_i}$ $\quad \alpha \cdot c_{i,j} + S_i$,
- processing the chunk on $\mathrm{N_i}$ $\quad \alpha \cdot w_{i,j}$,
- receiving the result from $\mathrm{N_i}$ $\quad \alpha \cdot c'_{i,j} + S'_i$.

We suggested in section 1 that the processing cost for a chunk should dominate its communication costs in a certain sense. We choose to formulate this assumption as:

$$\forall \alpha \in \mathbb{R}^{+*},$$
$$\alpha \cdot \min_{k \in \mathbb{N}^*} w_{i,k} \geq \left(\alpha \cdot \max_{k \in \mathbb{N}^*} c_{i,k} + S_i\right) + \left(\alpha \cdot \max_{k \in \mathbb{N}^*} c'_{i,k} + S'_i\right) \quad (2)$$
$$for\ i = 1, N.$$

Equation (2) ensures that the cost of sending chunks of any size α to a worker N_i and receiving the corresponding results is less than the processing time of these chunks.

The problem with the OL method is that computation never overlaps communication in any computing node, as the emission of the chunk of the next round is at best triggered by the return of the result of the previous one. Therefore, we adapted the OL method in order to be able to minimize these idle periods of the worker nodes. Moreover, we will state sufficient conditions to avoid these idle periods. If the nodes being used have been selected optimally during this phase, then the total duration of the execution of the application (makespan) is minimal.

4 The OLMR method

4.1 Overview of the method

The new method we present is based on the OL method. The purpose of our adaptation of the OL method is for

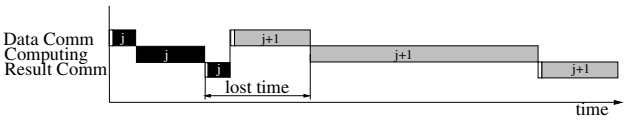

Figure 1. Worker node idle period between successive rounds with the OL method

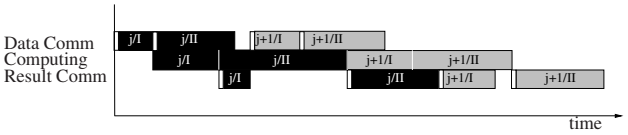

Figure 2. Overlapping between communication and computation with the OLMR method

worker nodes to avoid idle time with respect to computing. As can be seen in figure FIG.1, these inter-round starvations occur between the time instants when a worker begins to send the result for a round back to the supervisor and when it finishes the reception of the next round chunk. It has been well established by now that, when the total load is important compared to the available bandwidth between supervisor and workers, the workload should be delivered in multiple rounds [7, 25, 5]. Therefore we will have each worker receive its share of the load through multiple rounds. Our adaptation of the OL method draws its inspiration from the principle of the multi-installment strategy presented in [6]. So the new method we present is therefore denoted "OLMR method" (for On-Line Multi-Round method [9]) thereafter. The OLMR method divides the chunk sent to N_i for each round j into two subchunks "I" and "II" of respective sizes $\overline{\alpha}_{i,j}$ and $\alpha_{i,j} - \overline{\alpha}_{i,j}$. Dividing the chunks in two parts is enough in order to apply the principle, and the division allows the computation to overlap the communications between a worker and the supervisor as can be seen in figure FIG.2. In order to proceed, we compute $\alpha_{i,j}$ using the measurement of the elapsed time (including both communications and computation) for subchunk I of the previous round: $\overline{\sigma}_{i,j-1}$. We will show that, thanks to this anticipation (compared to the OL method) in the computation of $\alpha_{i,j}$, we can avoid the inter-round starvation.

Figure FIG.3 gives the scheduling algorithm of the OLMR method. The OLMR scheduler computes $\alpha_{i,j}$ in the same way as the OL scheduler does, and the values of $\sigma_{i,j-1}$ and $\overline{\alpha}_{i,j}$ as detailed later in the next subsections.

Paradoxically, while attempting to deal with the inter-round starvations inherent to the OL method, there is a risk of creating some intra-round starvation between subchunks I and II. We give below conditions to prevent both risks.

As we assume that (2) holds, intra-round starvation can be avoided if $\overline{\alpha}_{i,j}$ is large enough for the processing of subchunk I to overlap the sending of subchunk II (see an example of intra-round starvation on figure FIG.4).

while (the last data item has not been acquired) **do**
  if (Reception from N_i of the result of subchunk I of its $(j-1)^{th}$ round) **then**
    - Get $\overline{\sigma}_{i,j-1}$, $\omega_{i,j-1}$ and $c'_{i,j-1}$
    - Compute $\sigma_{i,j-1}$ ..... (cf. (8))
    - Compute $\alpha_{i,j}$ ..... (cf. (1))
    - Compute $\overline{\alpha}_{i,j}$ ..... (cf. (7))
    - Send a subchunk I of size $\overline{\alpha}_{i,j}$ to N_i
    - Send a subchunk II of size $(\alpha_{i,j} - \overline{\alpha}_{i,j})$ to N_i
  end if
end while

Figure 3. OLMR scheduler (On-Line Multi-Round)

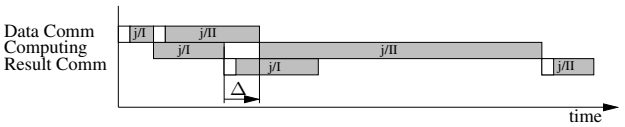

Figure 4. Example of intra-round starvation with the OLMR method

Let Δ be the duration of the idle period of N_i between the two subchunks of its j^{th} round, hence

$$\Delta = (S_i + (\alpha_{i,j} - \overline{\alpha}_{i,j}) \cdot c_{i,j}) - (\overline{\alpha}_{i,j} \cdot w_{i,j}).$$

There is no intra-round starvation if and only if $\Delta \leq 0$, that is

$$\overline{\alpha}_{i,j} \geq \frac{S_i + \alpha_{i,j} \cdot c_{i,j}}{w_{i,j} + c_{i,j}}. \quad (3)$$

Let us now go back to the risk of an inter-round starvation between the j^{th} and $(j+1)^{th}$ rounds of N_i, which could occur if subchunk I happens to be too large compared to subchunk II (see figure FIG.5). Let $\nu_{i,j}$ be some real number dominating $\overline{\alpha}_{i,j+1}$:

$$\nu_{i,j} \geq \overline{\alpha}_{i,j+1}.$$

Suppose that N_i is given a subchunk of size $\nu_{i,j}$ for its

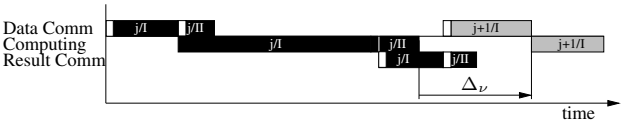

Figure 5. Example of inter-round starvation with the OLMR method

$(j+1)^{th}$ round, and let $\Delta_{\nu_{i,j}}$ be the duration of the corresponding idle period of N_i between its j^{th} and $(j+1)^{th}$ rounds. We have

$$\Delta_{\nu_{i,j}} = \left(S'_i + \overline{\alpha}_{i,j} \cdot c'_{i,j}\right) + \left(S_i + \nu_{i,j} \cdot c_{i,j+1}\right) - \left(\left(\alpha_{i,j} - \overline{\alpha}_{i,j}\right) \cdot w_{i,j}\right).$$

There is no inter-round starvation if and only if $\Delta_{\nu_{i,j}} \leq 0$, that is

$$\overline{\alpha}_{i,j} \leq \frac{\alpha_{i,j} \cdot w_{i,j} - \nu_{i,j} \cdot c_{i,j+1} - (S'_i + S_i)}{c'_{i,j} + w_{i,j}}. \quad (4)$$

It can easily be verified that if inequality (4) holds, then the necessary constraint

$$\overline{\alpha}_{i,j} < \alpha_{i,j}$$

holds too.

Relying on inequations (3) and (4), we can choose $\overline{\alpha}_{i,j}$ so as to avoid idle periods of N_i. We can derive different algorithms according to the way the value of $\nu_{i,j}$ is determined. Finally, nothing remains but to describe how $\sigma_{i,j-1}$ and $\overline{\alpha}_{i,j}$ are computed.

4.2 Determining $\overline{\alpha}_{i,j}$

The value of $\overline{\alpha}_{i,j}$ must be fixed according to constraint (4), which means that we need a value for $\nu_{i,j}$. We can decide such a value by extrapolating an upper bound for $\overline{\alpha}_{i,j+1}$ from the values of $\overline{\alpha}_{i,k}$ for the previous rounds, $\{\overline{\alpha}_{i,k}\}_{k=1,j-1}$. So long as inequalities (3) and (4) hold, an inaccuracy in the value of $\nu_{i,j}$ does not have any dramatic consequence on the course of the method. That is, if inequalities (3) and (4) are compatible, then starvation risks can be avoided.

As the amount of data processed during the steady-state phase is finite, there necessarily exists a real number λ_i ($\lambda_i \geq 1$) each N_i such that for:

$$\overline{\alpha}_{i,j+1} \leq \lambda_i \cdot \overline{\alpha}_{i,j} \qquad \forall j \in \mathbb{N}^*.$$

λ_i characterizes the amplitude of the fluctuations of $\overline{\alpha}_{i,k}$ between two successive rounds.

If λ_i can be estimated (see Remarks 2 and 3 for hints), then we have an upper-bound $\nu_{i,j}$ for $\overline{\alpha}_{i,j+1}$:

$$\nu_{i,j} = \lambda_i \cdot \overline{\alpha}_{i,j}. \quad (5)$$

The following Theorem proposes a way to set the value of $\overline{\alpha}_{i,j}$ so that constraints (3) and (4) are both satisfied.

Theorem 1 *Given $\alpha_{i,j}$, if $w_{i,j}$, $c_{i,j}$, S_i, $c'_{i,j}$ and S'_i satisfy (2) and*

$$\left(\alpha_{i,j} - (\lambda_i + 1)\right) \cdot w_{i,j} \geq \left(\lambda_i \cdot \alpha_{i,j} + (\lambda_i + 1)\right) \cdot c_{i,j} + (\lambda_i + 1) \cdot S_i \quad (6)$$

for i=1,N.
Then, taking

$$\overline{\alpha}_{i,j} = \frac{\alpha_{i,j}}{\lambda_i + 1}, \quad (7)$$

constraints (3) and (4) are satisfied. Therefore, the workers will compute without any idle period during the steady-state phase.

Proof: Thanks to (6), we have

$$\begin{aligned}\left(\alpha_{i,j} - (\lambda_i + 1)\right) \cdot (w_{i,j} + c_{i,j}) \geq & \left(\lambda_i \alpha_{i,j} + (\lambda_i + 1)\right) c_{i,j} + (\lambda_i + 1) S_i + \left(\alpha_{i,j} - (\lambda_i + 1)\right) c_{i,j}, \\ \frac{\alpha_{i,j}}{\lambda_i + 1} - 1 \geq & \frac{S_i + \alpha_{i,j} \cdot c_{i,j}}{w_{i,j} + c_{i,j}}.\end{aligned}$$

Then using the definition (7) of $\overline{\alpha}_{i,j}$, we have

$$\overline{\alpha}_{i,j} \geq \frac{S_i + \alpha_{i,j} \cdot c_{i,j}}{w_{i,j} + c_{i,j}}.$$

So constraint (3) is satisfied.

By definition (7), we have

$$\begin{aligned}\overline{\alpha}_{i,j} &\leq \frac{\alpha_{i,j}}{\lambda_i + 1}, \\ \lambda_i \cdot \overline{\alpha}_{i,j} \cdot w_{i,j} &\leq \left(\alpha_{i,j} - \overline{\alpha}_{i,j}\right) \cdot w_{i,j}.\end{aligned}$$

By hypothesis (2), this last inequality can be rewritten as:

$$\begin{aligned}\left(\lambda_i \cdot \overline{\alpha}_{i,j} \cdot c'_{i,j} + S'_i\right) + \left(\lambda_i \cdot \overline{\alpha}_{i,j} \cdot c_{i,j+1} + S_i\right) &\leq \left(\alpha_{i,j} - \overline{\alpha}_{i,j}\right) \cdot w_{i,j}, \\ \left(\overline{\alpha}_{i,j} \cdot c'_{i,j} + S'_i\right) + \left(\lambda_i \cdot \overline{\alpha}_{i,j} \cdot c_{i,j+1} + S_i\right) &\leq \left(\alpha_{i,j} - \overline{\alpha}_{i,j}\right) \cdot w_{i,j}.\end{aligned}$$

Using (5), we then obtain:

$$\left(\overline{\alpha}_{i,j} c'_{i,j} + S'_i\right) + \left(\nu_{i,j} c_{i,j+1} + S_i\right) \leq \left(\alpha_{i,j} - \overline{\alpha}_{i,j}\right) w_{i,j}.$$

Hence

$$\overline{\alpha}_{i,j} \cdot \left(w_{i,j} + c'_{i,j}\right) \leq \alpha_{i,j} \cdot w_{i,j} - \nu_{i,j} \cdot c_{i,j+1} - \left(S_i + S'_i\right).$$

That is, inequality (4) is satisfied. ■

Remark 1 *Parameters τ and λ_i are both characteristic of the evolution of the execution parameters. On the one hand, τ characterizes their speed of evolution. Practically, it is the period that should be used for reconsidering their value. On the other hand, λ_i measures the amplitude of their variations on such a period. The obvious dependence between τ and λ_i can take on the most varied forms. For instance, we can have rapid variations (small τ) of the execution parameters with little consequence on the scheduling of the application (λ_i close to 1), or on the contrary slow variations (large τ) with important consequences on the scheduling of the application (λ_i far from 1).*

Remark 2 *The knowledge of $\nu_{i,j}$ is implicitly the result of some extrapolation of the values $(\overline{\alpha}_{i,k})_{k=1,j}$ to get an upper-bound of $\overline{\alpha}_{i,j+1}$. If the variations are slight, one can use the quasi-stationary approximation of $\overline{\alpha}_{i,j+1}$ by $\overline{\alpha}_{i,j}$. In this case, we have*

$$\overline{\alpha}_{i,j+1} = \overline{\alpha}_{i,j}.$$

Then we only have to apply Theorem 1 with

$$\lambda_i = 1.$$

More generally, considering a polynomial interpolation of degree (p - 1) for the value of $\overline{\alpha}_{i,j+1}$, we have

$$\overline{\alpha}_{i,j+1} = p \cdot \overline{\alpha}_{i,j} - \sum_{k=1}^{p-1} \overline{\alpha}_{i,k}.$$

In this case, it suffices to apply Theorem 1 with

$$\lambda_i = p.$$

Remark 3 *Satisfying the hypotheses of Theorem 1 guarantees the absence of idle time for the workers but requires the knowledge of $(\lambda_i)_{i=1,N}$. Nevertheless, the OLMR method may still be used when these values (which characterize the dynamicity of execution parameters) are not known. Starting with arbitrary values (e.g. $\lambda_i = 1$ corresponding to a stability assumption) the scheduler could, if necessary, adjust λ_i values according to information provided by the workers at any round. Actually an inappropriate value of λ_i used for some round will lead to an intra- or inter-round starvation observable by the corresponding worker. The scheduler could then adjust this value for the next round, according to the type of starvation observed by the worker.*

Remark 4 *Although different, hypotheses (2) and (6) both make the assumption that processing should dominate communications. Recall that hypothesis (2) ensures an efficient usage of the supervisor-worker paradigm.*

4.3 Determining $\sigma_{i,j-1}$

In order to determine the size of the chunk to be sent for the next round without waiting for the result of the currently processed chunk, replacing the measured value $\sigma_{i,j-1}$ in expression (1) by some computed value derived from $\overline{\sigma}_{i,j-1}$ suffices. But we only know the values of the execution parameters for the data whose result have been received by the master. We choose to get these parameters just after the master has reveived the result for subchunck I of round $j-1$ (see tag "Snapshot" on FIG.6). It is another extrapolation problem. In order to solve it, we assume that the time taken by N_i to process some amount of data during its $(j-1)^{th}$ round is the same for both subchunks I and II.

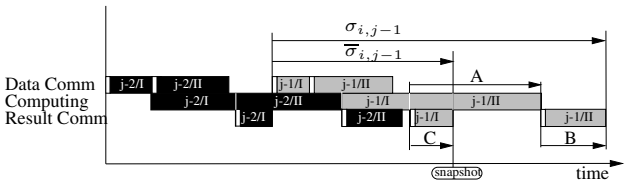

Figure 6. From the measurement of $\overline{\sigma}_{i,j-1}$ to the computation of $\sigma_{i,j-1}$

Using the notations introduced in figure FIG.6, and omitting the cost of the scheduling algorithm itself, we have:

$$\sigma_{i,j-1} = \overline{\sigma}_{i,j-1} + A + B - C,$$

$$\sigma_{i,j-1} = \overline{\sigma}_{i,j-1} + (\alpha_{i,j-1} - \overline{\alpha}_{i,j-1}) \cdot \omega_{i,j-1} + (\alpha_{i,j-1} - 2 \cdot \overline{\alpha}_{i,j-1}) \cdot c'_{i,j-1}. \quad (8)$$

Remark 5 *The values of $\omega_{i,j-1}$ and $c'_{i,j-1}$ can be estimated easily by the master with the help of N_i. There is no need to know the value of either the communication start-up times β_i and β'_i or that of $c_{i,j-1}$ in order to compute $\sigma_{i,j-1}$ by means of equation (8).*

4.4 Comparing OL and OLMR methods

In this section, we compare OL and OLMR methods and quantify the benefit of using OLMR compared to OL. We study their behaviour in identical settings: a static context.

Using the OLMR method requires that the hypotheses of Theorem 1 be satisfied. Lemma 6.1 in [12] sets the context of the OL method as static. In particular, due to the static nature of the execution environment, parameters $w_{i,j}$, $c_{i,j}$ and $c'_{i,j}$ do not depend on the round. Under these conditions, both methods send a chunk of the same size $\alpha_{i,j}$ to N_i for any round j; for the same value of τ. So processing a workload of size M by both methods requires the same number of rounds δ_M. Let us denote c_i, w_i and c'_i the value of $c_{i,j}$, $w_{i,j}$ and $c'_{i,j}$ for any round j. Here we estimate the gain γ_M of method OLMR over method OL when processing the same workload. Let $T_{OL}(M)$ and $T_{OLMR}(M)$ be the respective times for methods OL and OLMR to process a workload of size M.

$$\begin{aligned}
T_{OL}(M) &= \delta_M \cdot (S_i + S'_i) + M \cdot (c_i + w_i + c'_i), \\
T_{OLMR}(M) &= (S_i + S'_i) + \overline{\alpha}_{i,1} \cdot c_i + M \cdot w_i \\
&\quad + (\alpha_{i,\delta_M} - \overline{\alpha}_{i,\delta_M}) \cdot c'_i, \\
\gamma_M &= T_{OL}(M) - T_{OLMR}(M), \\
&= (\delta_M - 1) \cdot (S_i + S'_i) + (M - \overline{\alpha}_{i,1}) \cdot c_i \\
&\quad + (M - (\alpha_{i,\delta_M} - \overline{\alpha}_{i,\delta_M})) \cdot c'_i.
\end{aligned}$$

The gain γ_M is the direct consequence of overlapping computation and communications (see figure FIG.7).

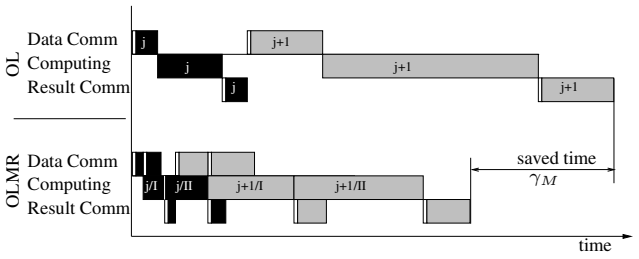

Figure 7. Comparison between OL and OLMR methods

5 Related work

The divisible load model (DLT) has been largely studied; it is the first model that enables optimality proofs for scheduling methods in the context we have chosen [23, 8, 20, 19]. This model is well suited to the problem of scheduling the clean-up phase. On the contrary, it is not suited to scheduling the steady-state phase, as the total workload is not known during this phase.

Several multi-round methods have been proposed in the literature [7, 25, 5]. On the one hand the iterated distributions of multi-round methods have the advantage (when using a one-port model without contention) of making the nodes active earlier, and on the other hand they have the drawback of increasing the time wasted by latencies (cf the affine cost). Several strategies for distributing the load to slaves have already been studied. First of all, some strategies fix the number of rounds arbitrarily (multi-installment methods [7]). They are well suited to a model with linear costs for homogeneous platforms. For heterogeneous platforms, other strategies have been proposed, which are able to take account of the affinity of costs when determining the load of the nodes for each round. For instance, the workload which is delivered at each round by the UMR method [25] follows a geometric progression whose common ratio is adjusted so that all the nodes work for the same duration in each round and so that computation overlaps communication exactly. It is proved [25] that this method minimizes the total execution time; provided that we know how to select the best set of nodes to be used. The PMR method [5] introduces periodicity for the rounds (without imposing any particular value for the period) and requires that all nodes work during the whole period and that computation overlaps communication exactly. It is proved [5] that this method maximizes the amount of load processed by time unit.

Unfortunately none of these methods can be used when the total workload is not known a priori. The OLMR method we have presented in this paper has the advantages inherent to multi-round methods: it takes account of both the heterogeneity of the platform and the affinity of costs. Moreover, it allows for the dynamicity of the execution parameters.

Parameter τ can be adjusted according to the finest time scale characterizing the evolution of the execution parameters. So doing, this evolution is taken account of (in average) over the duration of a round.

Under appropriate hypotheses (cf Lemma 6.1 in [12]), which are met when execution parameters are stable, rounds are asymptotically periodic (as for the PMR method). Be the hypotheses of Theorem 1 satisfied, the method definitely minimizes the idle time of the computing resources.

In the same way as the for PMR method [5], the use of the OLMR method we have presented must be coupled with some mechanism able to optimally select the resources to be used. Such resource selection can rely on heuristics, e.g. a greedy algorithm over the set of nodes ordered according to decreasing bandwidth (bandwidth-centric allocation [3]). It can be done at each round or according to the dynamicity of the execution parameters.

Under the hypotheses of Theorem 1 and if the choice of resources is optimal then the evaluation of this method by competitive analysis [18] with an off-line method is not necessary; due to the full use of the computing resources.

6 Conclusion

In this paper, we have defined a scheduling problem that we think is realistic when considering scheduling applications relying on data parallelism on shared resources. To the best of our knowledge, the scheduling problem we consider in this paper has not received much attention up to now. We have presented a new runtime scheduling method to optimize the distribution of tasks which can deal with the heterogeneity and dynamicity of the grid if our modelisation hypotheses are realistic; it can also be used when the information that scheduling algorithms traditionally need is lacking. Sufficient conditions have been stated for full usage of the resources by means of avoiding idle time.

In order to design the OLMR method, we had to consider the characterization of the dynamicity of the execution conditions. This led us to define $N + 1$ parameters: τ and $(\lambda_i)_{i=1,N}$ (see Remark 1). But the improvement made by OLMR to the on-line method presented in [12] has been quantified in a static execution context only.

This novel approach of scheduling problems is susceptible to numerous improvements and developments.

These developments are twofold: those tending to confirm the results obtained in this paper and those aiming at enlarging the potentialities of the On-Line Multi-Round method. First of all, it is useful to check experimentally that, under the hypotheses of our model, the method gives the expected results. For that, we are currently developping simulation programs, using the SimGrid toolkit [10] in

order to study the behavior of the OLMR method in various conditions and make comparisons with other methods. When this step has been passed successfully, we can then test the efficiency of the method in the framework of the AIPE (**A**utomatic **I**ntegration for **P**arallel **E**xecution) [17] project. AIPE is a software production chain which enables end-users to get parallel execution of an application on grids without writing anything more than its sequential code. It produces automatically a parallel code relying on the supervisor-worker paradigm for which scheduling plays an important role. The experimentation with AIPE would allow a validation of the hypotheses of the model.

The new method could be adapted in different ways that we are going to discuss now. In this paper, τ and λ_i have implicitly been considered as constant throughout all the rounds. This hypothesis restricts the degree of approximation (order one) of the dynamicity that the scheduler takes into account. From one round to the next, the value of τ could be adapted in order to take account of the evolution of heterogeneity and dynamicity that would be noticed.

References

[1] D. Altilar and Y. Paker. An optimal scheduling algorithm for parallel video processing. In *Proceedings of the International Conference on Multimedia Computing and Systems*, IEEE Computing Society Press, 1998.

[2] O. Beaumont. Nouvelles méthodes pour l'ordonnancement sur plates-formes hétérogènes. Habilitation à diriger des recherches, Université de Bordeaux 1 (France), December 2004.

[3] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent task on heterogeneous platform. Technical Report 4210, INRIA, Rhône-Alpes, Grenoble(France), June 2001.

[4] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent task on heterogeneous platform. In *Proceeding of the 16th International Parallel and Distributed Processing Symposium (IPDPS'02)*, IEEE Computing Society Press, April 2002.

[5] O. Beaumont, A. Legrand, and Y. Robert. Optimal algorithms for scheduling divisible workloads on heterogeneous systems. Technical Report 4595, INRIA, Le Chesnay(France), October 2002.

[6] V. Bharadwaj and G. Barlas. Efficient scheduling strategies for processing multiple divisible loads on bus networks. *Journal of Parallel and Distributed Computing*, 62(1):132–151, January 2002.

[7] V. Bharadwaj, D. Ghose, and V. Mani. Multi-installment load distribution in tree networks with delays. *IEEE Transactions on Aerospace and Electronic Systems*, 31(2):555–567, 1995.

[8] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. Scheduling divisible loads in parallel and distributed systems. *IEEE Computing Society Press*, 1996.

[9] S. Boutammine, D. Millot, and C. Parrot. A multi-round online scheduling method for grid computing. Technical Report 05008INF, INT/GET, Évry(France), September 2005.

[10] H. Casanova, A. Legrand, and L. Marchal. Scheduling distributed applications: the simgrid simulation framework. In *Proceedings of the 3th International Symposium on Cluster Computing and the Grid (CCGrid03)*, IEEE Computing Society Press, 2003.

[11] L. Dong, V. Bharadwaj, and C. Ko. Efficient movie retrieval strategies for movie-on-demand multimedia services on distributed networks. *Multimedia Tools and Applications*, 20(2):99–133, 2003.

[12] M. Drozdowski. *Selected problems of scheduling tasks in multiprocessor computing systems*. PhD thesis, Instytut Informatyki Politechnika Poznanska, Poznan, 1997.

[13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal Supercomputer Applications*, 25(3), 2001.

[14] A. Legrand, Y. Yang, and H. Casanova. Np-completeness of the divisible load scheduling problem on heterogeneous star platforms with affine costs. Technical Report CS2005-0818, UCSD/CSE, March 2005.

[15] Y. Liu. Grid scheduling. University of Iowa, Department of Computer Science, http://www.cs.uiowa.edu/~yanliu/QE/QEreview.pdf, 2004.

[16] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. Steady-state scheduling of multiple divisible load applications on wide-area distributed computing platforms. *Int. Journal of High Performance Computing Applications*, 2006, to appear.

[17] D. Millot, C. Parrot, and E. Renault. Transparent usage of grids for data parallelism. In *Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems (PDCS05)*, 2005.

[18] K. Pruhs, J. Sgall, and E. Torng. *Online scheduling, Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. J. Leung, ed., CRC Press, 2004.

[19] T. Robertazzi. Divisible load scheduling. http://www.ece.sunysb.edu/~tom/dlt.html.

[20] T. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5)(63-68), 2003.

[21] T. Robertazzi, J. Sohn, and S. Luryi. Load sharing controller for optimizing monetary cost, March 30 1999. US patent # 5,889,989.

[22] J. Schopf. Ten actions when grid scheduling: the user as a grid scheduler. *Grid resource management: state of the art and future trends*, pages 15–23, 2004.

[23] J. Sohn, T. Robertazzi, and S. Luryi. Optimizing computing costs using divisible load analysis. *IEEE Transactions on Parallel and Distributed Systems*, 9(3), March 1998.

[24] K. van der Raadt and H. C. Y. Yang. Practical divisible load scheduling on grid platforms with apst-dv. In *Proceeding of the 19th International Parallel and Distributed Processing Symposium (IPDPS'05)*, volume 1, page 29b, IEEE Computing Society Press, April 2005.

[25] Y. Yang and H. Casanova. *UMR: A Multi-Round Algorithm for Scheduling Divisible Workloads*. IEEE Computing Society Press, April 2003.